UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS

SECCION DE POSGRADO

COMPARACIÓN DE DOS PROCESOS DE DESARROLLO DE SOFTWARE USANDO LOS MÉTODOS ICONIX Y XP, CASO: COMERCIALIZACIÓN DE LA TARA EN LA REGIÓN AYACUCHO

TESIS

Para Optar el Grado Académico de:
Maestro en Ciencias con Mención en
INGENIERIA DE SISTEMAS

Ing. PORRAS FLORES, EFRAIN ELIAS

LIMA – PERU

2010

DEDICATORIA

A Dios y Mis Padres por haberme dado la vida e inteligencia para lograr Mis metas y contribuir al Desarrollo Social del Perú.

AGRADECIMIENTO

A Mi Alma Mater la Universidad Nacional de Ingeniería, a mis Maestros de la Sección de Posgrado en Ingeniería de Sistemas y a todas las personas que hicieron posible la culminación de la presente investigación.

CONTENIDO

RESUMEN

Desarrollar software aplicando las metodologías ICONIX, XP y evaluar la calidad del producto software, para PYMES que tienen escasos recursos humanos, financieros e infraestructura es una problemática vigente. El objetivo es desarrollar un producto software, usando ICONIX y XP, luego medir, evaluar y comparar la calidad de los productos software.

El proceso ICONIX descrito considera sus fases e hitos para cada uno, sus tareas, artefactos, técnica y responsables para el análisis de requisitos, diseño preliminar, arquitectura técnica, diseño e implementación y sus revisiones. El proceso XP esta formulado para las fases de exploración, planificación e iteración y, sus tareas, artefactos, técnica y responsables. Para evaluar la calidad del producto software se procedió a ponderar características, sub características, calificación: a nivel de atributo, por sub característica, ponderada por característica, ponderada total e interpretación del grado de calidad del producto. Las medidas de la calidad del producto presentan, dato cuantitativo proporcional, usamos la técnica estadística Ji-cuadrado, distribución multinomial, para definir cual de las metodologías ICONIX o XP produce software con mejor calidad de producto.

El control y aseguramiento de calidad de los productos software desarrollados con ICONIX y XP, se realizó con normas técnicas aceptadas y estandarizadas en el Perú. El análisis estadístico para la calidad interna del software desarrollado con ICONIX y XP, indica que son estadísticamente iguales, la metodología ICONIX para la calidad externa es estadísticamente superior a la

metodología XP. El análisis de la calidad del producto software, para el proceso ICONIX presenta nivel obtenido (74.07%) y es superior al nivel requerido (66.21%), para el proceso XP el nivel obtenido (71.98%) es superior al nivel requerido (66.21%). La prueba estadística indica que no existe diferencia significativa entre la calidad del producto desarrollado con ICONIX y XP.

PALABRAS CLAVE

Software, ICONIX, XP, Programación Orientada a Objetos, UML, Comercialización de Tara.

INTRODUCCION

Comercializar tara en la Región Ayacucho, es una actividad productiva de recolección, con incipientes técnicas de manejo agronómico, producción de tara dispersa e individualizada, intervención de instituciones facilitadoras estatales y privadas desorganizada, informalidad en el mercado con efectos en la variación de precios de producción y comercialización.

La programación extrema (XP) esta dirigido a grupos pequeños y medianos de construcción de software, con requisitos poco definidos que cambian rápidamente. El código en XP es el principal artefacto y el aseguramiento de la calidad se centra en el código, asegurar la calidad de las historias de usuario se logra con la participación del cliente (Ariel y Bastarrica, 2005). En una encuesta sobre XP, informan que existe riesgo alto en la falta frecuente del cliente y la programación por pares. Los aspectos positivos son propiedad colectiva del código, programación en pareja y enfoque de XP sobre objetivos del cliente (Rumpe y Schröder, 2001).

ICONIX es considerada una metodología pura, práctica y simple, que realiza análisis y diseño, con la característica de "rastreabilidad de los requisitos", con los entregables, modelo de dominio, modelo de caso de uso, diagrama de robustez, diagrama de secuencia y diagrama de clases. (Pantoja, 2009).

Existen modelos de calidad para productos software, la mayoría basados en la norma ISO 9126, las métricas validadas teóricamente son 3 % y empíricamente 37 % (Esparza, 2006). El proceso de gestión de calidad del software, tiene tareas como: control de calidad del software y asegurar la calidad del software

(Scalone, 2003). En una encuesta sobre la calidad del desarrollo de sistemas, opinan que medir la calidad del producto software en desarrollo y las características de calidad como funcionalidad y usabilidad son las más importantes (Santana, 2003).

CAPITULO I
PLANTEAMIENTO DE LA INVESTIGACION

1.1 DIAGNOSTICO Y ENUNCIADO DEL PROBLEMA

Durante las últimas décadas se ha desarrollado software usando diversas metodologías, particularmente a partir de 1990, se utiliza el concepto, calidad del producto software, mediante la evaluación de indicadores de calidad del producto, hoy existen muchos métodos y estándares para medir la calidad del producto software de acuerdo a la realidad concreta de cada proyecto, debemos determinar la metodología mas apropiada para desarrollar software aplicando los procesos ICONIX y XP, evaluando y comparando el desarrollo mediante los indicadores de calidad del producto, luego seleccionar el proceso mas adecuado, para lo cual usamos el caso "Comercialización de la Tara en la Región Ayacucho". Algunas metodologías actuales tienen ventajas y limitaciones, que son:

a. La curva de aprendizaje del XP, SCRUM, FDD e ICONIX es rápida, la del proceso unificado es lenta.

b. ICONIX aporta mejor documentación de modelado al ser comparada con otros métodos ágiles.

c. El proceso unificado es una metodología rigurosa y disciplinada pero causa parálisis durante el desarrollo del software y es aconsejable para proyectos grandes con equipos amplios para el desarrollo.

d. XP depende mucho de la especialización de los programadores, al igual que otras metodologías ágiles, necesita la participación del cliente.

e. SCRUM es bueno para la administración de proyectos pero fracasa en las demás etapas siendo obligatorio combinarla con otros métodos.

f. ICONIX presenta acceso minimalista, es más aplicable a proyectos de pequeño o mediano alcance y con recursos limitados.

g. Utilizando los métodos anteriores podemos aplicar patrones durante la etapa de diseño como una forma de garantizar la calidad del código.

EXTREME PROGRAMMING (XP)

La filosofía de XP es lograr la excelencia en el desarrollo con equipos hábiles, no es aconsejable para equipos nuevos, poco integrados o desbalanceados, en cuanto a la calidad técnica y de cooperación de sus integrantes. XP no cuenta con un diseño riguroso y disciplinado, sin embargo algunas de sus técnicas resaltan la cooperación e integración de los equipos de trabajo alrededor de las metas comunes, esto siempre será bienvenido en un proyecto software. XP aboga por aplicar un conjunto de buenas prácticas de programación que son producto de la experiencia de muchos años de algunos de los mejores programadores existentes. La aplicación de este cúmulo de prácticas no tiene otro tipo de resultado que mejorar el rendimiento y la calidad del código resultante. Sin embargo, se nota cierto elitismo y facilismo para obviar las probadas prácticas de metodologías de desarrollo de software, por lo que son recomendables para equipos balanceados y con altos niveles técnicos, además de primar la colaboración entre sus integrantes. La falta de un diseño riguroso limita la capacidad de producción de buena documentación técnica para el mantenimiento del software. (Stephens y Rosenberg, 2003)

ICONIX

Es un proceso minimalista que trata de evitar la parálisis del análisis, como ocurre con otras metodologías de ingeniería de software como el proceso unificado, ICONIX es un modelo simple que puede ser instanciado como una mejora del proceso unificado. El proceso ICONIX está entre el proceso unificado y el enfoque XP. Dirigido por casos de uso como el proceso unificado, pero no posee toda la sobrecarga de modelado del proceso unificado. Es relativamente pequeño como XP, pero no descarta las notaciones de análisis y diseño como lo hace XP. El enfoque ICONIX es minimalista y esta focalizado entre los casos de uso y el código. Uno de los aspectos negativos que tiene ICONIX es su escasa visión sobre las etapas de prueba del software, para esto se ha recomendado combinar ICONIX con Test-Driven Development (TDD), especialmente a través del uso de las xUnit de java. (Rosenberg et al., 2005)

CARACTERISTICAS DEL SISTEMA PARA COMERCIALIZAR TARA

La producción de tara en Ayacucho se estima en 810 Ha el año 2007, 35% instaladas y 65% silvestres. La extensión manejada de tara es 57%, representa US $ 4.76 millones en exportaciones, el año 2006 se ha producido 5083 TM, en agosto del 2007 el precio promedio de tara en chacra fue 3.20 soles por Kg. el mas alto de la historia. El productor realiza labores agronómicas, de cosecha y venta al acopiador. El acopiador compra la tara en vaina de los productores, almacena en sus centros de acopio y vende a un transformador de la Región. La comercialización de tara comienza en abril y termina afines de agosto. Los adelantos por compras y otras transacciones se realizan de manera informal, es decir, no existen contratos, recibos ni control de balanzas, los adelantos se cobran en un período no mayor a un mes. El transformador compra la tara a los acopiadores, previa revisión, mínimo 100 Kg., con precios mayores al que se paga al productor, trasladando los productos en camiones de recolección. El año 2006 el 93.8% de la producción de tara provenía de las provincias de Huanta y Huamanga, 3.8% de Cangallo y La Mar y 2.4% de otras provincias. El transformador traslada el producto por sus medios, como mínimo cargas de 10 TM. previa revisión de calidad. En Huanta y Huamanga se paga mas por la tara que en otras provincias. Las provincias de Cangallo y La Mar, tienen precios mas bajos que las primeras, pero aún altos respecto a otras provincias. De las 5083 TM de tara, el 74% se comercializa en vaina y el 26% transformado en goma y harina. Los principales empresas demandantes del mercado Ayacuchano son: Silvateam Perú 35%, Productos del País 30%, Exportadora el Sol 8% y otros 27%. Las principales empresas exportadoras del país son: Silvateam Perú 33.7%, Exportadora el Sol 17,7%, Transformadora Agrícola 10.7%, Productos del País 9%, Molinos Chipoco 9%, Exportaciones de la Selva 7% y otras empresas 12%. (Avendaño et al., 2007)

Según De la Cruz (2004), nuestro país exporta la tara como dos derivados, la harina de tara, sub partida nacional 1404.10.30.00 y goma de tara, sub partida nacional 1302.39.10.00. La harina de tara es utilizada para la obtención de varios productos entre los que destaca el ácido tánico, empleado para curtiembres y los mucílagos de semillas de tara se exportan como goma de tara, utilizada en la industria alimenticia. Ambos sub - productos tienen un

enorme potencial que debe ser aprovechado, para ello, debería promocionarse los resultados de los estudios técnicos que se realicen y diseñar una efectiva estrategia de producción, transformación y comercialización de los derivados de la tara.

EL PLAN ESTRATÉGICO REGIONAL EXPORTADOR - REGIÓN AYACUCHO

El análisis FODA realizado sobre la competitividad en relación a la tara, en el taller de planeamiento estratégico llevado a cabo los días 6, 7 y 8 de julio del 2005 en Ayacucho y validado el 16 de agosto del mismo año, por actores de la cadena productiva de la tara, considera lo siguiente (MINCETUR, 2005):

F1	Ayacucho tiene productos con ventajas a nivel mundial (tara, barbasco, vicuñas y cochinilla)
F12	La tara producida en Ayacucho tiene excelente concentración de taninos, favorecida por suelos calcáreos y pocas enfermedades que afectan este cultivo
F15	Masivo desarrollo de Internet con acceso a precios módicos
F20	Crecimiento del número de cadenas y asociaciones de productores

Tabla N° 1.1: Fortalezas del análisis FODA.

D2	Altos índices de pobreza y extrema pobreza
D10	Capital de las unidades productivas orientadas al comercio exterior es escaso
D32	La exportación es una actividad relativamente desconocida por la mayoría de los agentes económicos de la Región
D33	La investigación no está articulada con la realidad productiva
D36	La organización de las cadenas productivas es débil
D39	Los niveles de productividad y calidad en todos los sectores son bajos y de poca competitividad
D41	No hay articulación entre la programación de cultivos y el mercado
D45	Poca inversión e integración tecnológica, resultantes en un bajo desarrollo tecnológico
D49	Se carece de información sobre la demanda de los productos exportables en la Región y de los mercados internacionales

Tabla N° 1.2: Debilidades del análisis FODA.

O1	El TLC con EE.UU. facilitará la colocación de productos en el mercado norteamericano
O4	La voluntad política del gobierno nacional por promover las exportaciones
O7	El establecimiento de acuerdos de libre comercio con otros países

Tabla N° 1.3: Oportunidades del análisis FODA.

A6	Plagas y enfermedades que afecten a los cultivos
A8	Aparición de fenómenos climáticos extremos
A12	La desarticulación del estado con los sectores productivos
A13	La firma de un tratado de libre comercio mal negociado

Tabla N° 1.4: Amenazas del análisis FODA.

OBJETIVOS ESTRATEGICOS

Luego de hacer el cruce entre los factores internos y externos de acuerdo a las tablas 1.1, 1.2, 1.3 y 1.4 respecto a sistemas de información, se listan algunas matrices del plan estratégico exportador de la Región Ayacucho, como sigue:

OBJETIVO ESTRATÉGICO 2: Diversificar y consolidar la presencia de las empresas, productos y servicios peruanos en los mercados de destino priorizado			
ACTIVIDAD 2: Facilitar el acceso de la clase empresarial a información sobre mercado			
Indicador de logro:	Enlazar la oferta regional exportable con la demanda Internacional		
Indicador de avance:	Desarrollo de un sistema de información comercial internacional a finales de 2006.		
TAREA	**META**	**INDICADORES**	**RESPONSABLES**
T1: Recopilar información existente de la demanda internacional de productos exportables de la Región	Conocer el comporta-miento del mercado	• A finales del año 2006, conocer un 30% del mercado internacional de productos exportables de la región. • En el 2008, incremento del 60% del mercado. • En el 2010 conocer el 100% del mercado.	Gobierno Regional CERX RREE DIRCETUR-MINCETUR
T2: Implementar un sistema de Información sobre la oferta local exportable, demanda internacional, precios, condiciones de acceso a mercados, etc., de acuerdo a la necesidad regional y articulada con el SIICEX.	Contar con un sistema de Información actualizado	• En el 2006 se establece un convenio con PROMPEX para intercambio de información. • A fines de 2006 el sistema está disponible al público.	Centro de Competiti-vidad PROMPEX Gobierno Regional CERX

Tabla N° 1.5: Objetivo Estratégico N° 2, Actividad N° 2 y Tareas N° 1 y 2.

De acuerdo a los objetivos estratégicos, actividades, tareas, metas e indicadores, presentados en la tabla 1.5, las metas no se han cumplido, particularmente la tarea 2, su meta y el indicador que para fines del año 2006 se debe tener el sistema de información para comercialización disponible al público, realidad que no existe, persistiendo el problema.

En cuanto a información que entregan algunas instituciones, estas son distribuidas a mediano y largo plazo a los actores de la cadena en la Región Ayacucho, como el MINAG – Ayacucho, INIEA, UNSCH, entre otros.

La información no esta ha disposición del productor, acopiador, comercializador, transformador y mercado, el modelo económico exige contar con información que permita caminar hacia la competitividad, esto ha abierto expectativas en actores que están más y mejor informados sobre los precios, producción, costos, épocas de siembra y cosecha, mercado, experiencias exitosas, entre otros. Sin embargo, hay poca información disponible, en términos de confiabilidad y accesibilidad, por otro lado, lo más importante, existe sectores de la población de zonas rurales excluidas de la información.

1.2 DEFINICION DEL PROBLEMA DE INVESTIGACION

PROBLEMA PRINCIPAL

¿Cómo desarrollar software aplicando las metodologías ICONIX y XP y comparar los atributos de calidad del producto, con las características de ser PYMES, contar con escasos recursos humanos, financieros e infraestructura, para apoyar la comercialización de la tara en la Región Ayacucho?

PROBLEMAS SECUNDARIOS

a. ¿Cómo determinar las características y los atributos de calidad del producto a medir para desarrollar el software que apoya la comercialización de la tara en la Región Ayacucho?
b. ¿Cómo hacer un proceso de desarrollo de software que apoya la comercialización de la tara en la Región Ayacucho, aplicando la

metodología ágil XP?

c. ¿Cómo hacer un proceso de desarrollo de software que apoya la comercialización de la tara en la Región Ayacucho, aplicando la metodología disciplinada ICONIX?

d. ¿Cómo comparar los dos procesos de desarrollo de software y sus resultados bajo los atributos de calidad determinados?

1.3 DELIMITACION DE LOS OBJETIVOS DE INVESTIGACION

1.3.1 OBJETIVO GENERAL

Desarrollar el software aplicando las metodologías ICONIX y XP y comparar los atributos de calidad del producto para apoyar la comercialización de la tara en la Región Ayacucho, con las características de ser una PYME, contar con escasos recursos humanos, financieros e infraestructura.

1.3.2 OBJETIVOS ESPECIFICOS

a. Determinar las características y los atributos de calidad del producto a medir para desarrollar el software que apoya la comercialización de la tara en la Región Ayacucho.

b. Elaborar un proceso de desarrollo de software que apoya la comercialización de la tara en la Región Ayacucho, aplicando la metodología ágil XP.

c. Elaborar un proceso de desarrollo de software que apoya la comercialización de la tara en la Región Ayacucho, aplicando la metodología disciplinada ICONIX.

d. Comparar los dos procesos de desarrollo de software y sus resultados bajo los atributos de calidad que deben estar alineados a los requerimientos del negocio.

e. Obtener un software de calidad que apoya la comercialización de la tara en la Región Ayacucho.

1.4 HIPOTESIS DE LA INVESTIGACION

Para desarrollar software aplicando el método formal ICONIX se

obtendría mejores resultados de calidad del software (producto) que aplicando el método ágil XP, bajo las mismas condiciones, con las características de ser una PYME, contar con escasos recursos humanos, financieros e infraestructura, que apoye la comercialización de la tara en la Región Ayacucho.

1.5 JUSTIFICACION Y DELIMITACION DE LA INVESTIGACION

1.5.1 IMPORTANCIA DEL TEMA

IMPORTANCIA TÉCNICA

Se plantea formular un procedimiento para desarrollar software considerando una metodología formal como ICONIX y otra ágil como XP, para desarrollar software urgente y sostenible, aplicable a proyectos pequeños y medianos, usando para esto el caso, comercialización de tara en la Región Ayacucho, evaluando el producto a través de los indicadores de calidad y luego seleccionar el mejor. Dicho proceso puede ser usado en otra Región del Perú, para la comercialización de otras cadenas productivas, usando tecnologías de información actuales y sin costo de implementación, por ser licencias de software libre, el software se ejecutará en entorno Web con tecnologías de Internet que permitirán su funcionamiento continuo.

IMPORTANCIA SOCIOCONÓMICA PARA LOS ACTORES DE COMERCIALIZACION

Los actores internos y externos para comercializar tara en la Región Ayacucho, disponen de información en línea que esta en la Web, los beneficios indirectos esperados socio económicamente por apoyar la comercialización de tara en las provincias de Huanta, Huamanga, La Mar, Cangallo y Víctor Fajardo de la Región Ayacucho, permitirán mejorar los precios de venta de tara en vaina del productor, los acopiadores dispondrán de oportunidades para comprar tara en vaina debido a la oferta que existe en la Web, la comercialización de tara y derivados presenta mejores productos y precios, los transformadores obtienen mejores mercados para los derivados de la tara de exportación, los proveedores de bienes y servicios de la cadena productiva de tara identifican a sus clientes y comercializan su bienes o servicios.

1.5.2 JUSTIFICACION

La industria nacional del software es incipiente, presentando escasa oferta de software para cadenas productivas, se requiere formular un proceso de desarrollo de software que incorpore una metodología e indicadores de calidad del producto software, que sirva como referencia para proyectos pequeños y medianos de desarrollo de software, porque el País tiene miles de negocios con estas características, tomamos el caso comercialización de tara en la Región Ayacucho, que no cuenta con información oportuna y de calidad para la toma de decisiones del nivel operativo y táctico de los actores internos y externos de la cadena productiva. Por otra parte, la comunidad académica necesita disponer de procesos completos para desarrollar software e incluirlos en la formación de ingenieros de sistemas o informáticos.

1.5.3 DELIMITACION

La investigación debe identificar y formular el proceso para desarrollar software usando una metodología formal y otra ágil, existentes y mas usadas, para el proceso de desarrollo propuesto debe considerase las características de ser una PYME, contar con escasos recursos humanos, financieros e infraestructura para el desarrollo de software urgente, ser una pequeña o medina aplicación, ser evaluado con los mismos indicadores de calidad del producto, con esta perspectiva se tomara el caso: Comercialización de tara para la Región Ayacucho, considerado en el Plan Estratégico del Ministerio de Comercio Exterior – Ayacucho.

CAPITULO II
MARCO TEORICO

2.1 ANTECEDENTES

En nuestro País no existe trabajos sobre comparación de procesos de desarrollo de software, para proyectos pequeños y medianos, mediante los atributos de calidad del producto software y, evaluados utilizando métricas, razón por la que se referencia bibliografía externa.

2.1.1 LA CALIDAD DEL PRODUCTO SOFTWARE

Para la variable "calidad del desarrollo de sistemas", Santana (2003) realizó una encuesta, evaluando por importancia: 1-baja, 2-media, 3-alta, y 4-muy alta. Los encuestados, dicen que los factores de calidad importantes son la fiabilidad y funcionalidad, la razón mas importante para medir el software es para indicar la calidad del producto en desarrollo, y el factor que más incide en la productividad es el humano, como vemos tabla 2.1.

		Importancia					
		1	2	3	4	**Promedio**	**Moda**
Factores de calidad en el Software	Confiabilidad	1	2	8	17	3.5	4
	Eficiencia	1	2	19	6	3.1	3
	Funcionalidad	0	3	7	18	3.5	4
	Mantenibilidad	2	7	13	6	2.8	3
	Portabilidad	7	13	5	3	2.1	2
	Usabilidad	2	3	15	8	3.0	3
Razones para medir el software	Indicar la calidad del producto en desarrollo	1	3	10	14	3.3	4
	Evaluar la productividad del personal involucrado	1	7	17	3	2.8	3
	Analizar uso de métodos de ingeniería de software	0	8	12	8	3.0	3
	Establecer línea base para futuras estimaciones	5	7	12	4	2.5	3
Factores Involucrados	Humanos	1	0	12	17	3.5	4
	Del problema	0	10	14	4	2.8	3
	Del proceso	0	2	14	12	3.4	3
	Del producto	1	8	16	3	2.8	3
	De recursos	3	5	15	5	2.8	3

Tabla N° 2.1: Promedios para los factores de productividad y calidad. (Santana, 2003)

Las razones cuestionadas fueron: evaluar la productividad del personal, analizar métodos de ingeniería del software, y establecer la línea base para futuras estimaciones, como se muestra a la figura 2.1.

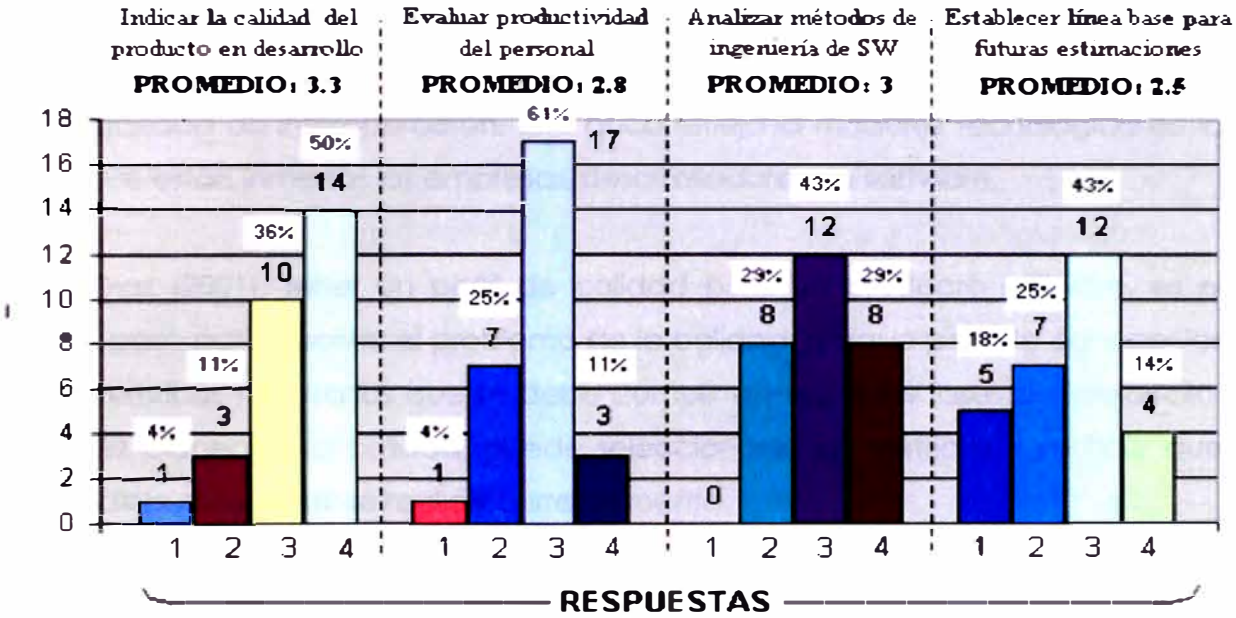

Figura N° 2.1: Razones para medir el software. (Santana, 2003)

De acuerdo a Díaz (2001), el aseguramiento de la calidad (SQA), es una actividad necesaria en la ingeniería del software, el perfil de calidad de un producto software ayudará a planear la calidad del producto, actividad subvalorada. Aplicó una encuesta a 45 empresas, programadores y líderes de proyectos, obteniendo los siguientes resultados:

a. El 84% de los encuestados opina que el aseguramiento de la calidad de sus productos es muy importante, de éstos, sólo el 57% tiene algún procedimiento para hacer la tarea;
b. Las empresas que persiguen el aseguramiento de la calidad en sus productos desde la planeación son el 73.7%, existe esfuerzo para asegurar la calidad del producto;
c. El 57% opina tener algún procedimiento para asegurar la calidad, sólo el 15% tiene una metodología específica, el resto hace esfuerzos para lograr la calidad de sus productos, pero de manera no disciplinada y rigurosa;
d. El 53% no contesto la pregunta, si han tenido resultados sus esfuerzos para asegurar la calidad del producto, sus resultados no son muy favorables, solo el 5% lo admite abiertamente;

e. Las características de calidad más importantes fueron, la fiabilidad y usabilidad, como producto final. La principal preocupación de los desarrolladores es la "primera impresión" que tiene un usuario cuando comienza a trabajar con un software. La causa podría ser que el usuario no puede usar algo o que los resultados no son los esperados;
f. Las características de calidad menos importantes son la portabilidad y facilidad de interoperación. Esto quizá refleja la madurez tecnológica en la que están inmersos las empresas/desarrolladores de software.

Para Díaz (2001), tener un perfil de calidad para un producto software, es el primer paso para resolver el problema de la calidad, porque permite conocer las características necesarias que se debe cumplir en todas las fases del desarrollo. Una vez planeada la calidad, puede seleccionarse las métricas y verificar que cada atributo a medir se realice correctamente.

Martín (2004), indica que la ingeniería de software, al igual que otras ingenierías, necesita información de métricas e indicadores para poder especificar, predecir, evaluar y analizar distintos atributos y características de los productos y procesos, que participan en el desarrollo y mantenimiento del software, estas métricas e indicadores debe ser compartida, reusada, interpretada y aplicada a distintos ambientes de desarrollo y mantenimiento. El modelo de calidad del software debe proporcionar una definición formal de conceptos consensuados que aseguren la interpretación correcta del conocimiento.

Según Scalone (2003), la gestión de la calidad del software (producto y proceso) esta formada por; planificación de la calidad (establecer los objetivos de calidad y especificar los procesos), control de calidad del software (hacer pruebas de software para encontrar errores), aseguramiento de la calidad del software (evaluar las características del software usando métricas para cuantificar los resultados obtenidos del software), mejora de la calidad (auditar el cumplimiento de las características del software y determinar posibles mejoras o cambios). La calidad del producto, presenta diferentes modelos y estándares con sus características, las que tienen asociadas sub características y métricas, donde la evaluación puede generar cambios en el desarrollo del software. Las razones

para medir el software son: indicar la calidad del producto, evaluar la productividad del personal que desarrolla, evaluar los beneficios (en términos de productividad y calidad) al usar nuevos métodos y herramientas de ingeniería de software, establecer una línea base de estimación.

Las métricas deben seleccionarse en función a los objetivos del negocio en particular Scalone (2003), los desarrolladores tienen objetivos comunes como: respetar el presupuesto y los plazos, minimizar los errores antes y después de la entrega del producto e intentar mejorar la calidad y la productividad. Las métricas deben ayudar al diseño de casos de prueba, las métricas de producto son privadas para un desarrollador y pueden servir para desarrollar métricas de proyecto que sean públicas para un equipo de software, enfocadas a predecir y controlar:

a. El tamaño (líneas de código, bytes de código, operadores y operandos);
b. La estructura (control de flujo, relación entre componentes, cohesión y acoplamiento);
c. La complejidad (combinación de tamaño y estructura);
d. Los indicadores para controlar la documentación;
e. La estabilidad (los cambios incrementan el número de fallas).

Esparza (2006), manifiesta que existen varios modelos de calidad para procesos y productos software, la mayor parte basados en la norma ISO 9126, modelo de calidad del producto software que incluyen calidad interna, calidad externa y calidad en uso, para la calidad interna y externa define seis características de calidad, que se dividen en sub características, que a su vez están descompuestas en atributos, cuyos valores se calculan usando métricas. El modelo Quint2 es una ampliación de ISO 9126, Quint2 distingue seis características principales y 32 sub características apropiadas para productos Web. Asimismo, clasificó 385 métricas, donde el 44 % están relacionadas a la presentación y 48 % a la usabilidad, sobre la validación de métricas el 3 % están validadas teóricamente y el 37 % son validadas empíricamente.

Para Diez (2003), muchos profesionales no aseguran la calidad del software (SQA), por las siguientes razones:

a. Los responsables del desarrollo evitan tener costos adicionales y no prevén los beneficios a largo plazo;
b. Muchos desconocen y creen que hacen lo correcto respecto al SQA;
c. Pocos saben dónde ubicar el SQA dentro de la organización;
d. Todos quieren evitar alguna burocracia que el SQA tiende a introducir en el proceso de la ingeniería del software.

Los beneficios esperados, por aplicar formalmente el SQA, son:

a. El software tendrá menos errores, por tanto, se reducirá el esfuerzo y tiempo de pruebas y mantenimiento;
b. Se genera mayor fiabilidad, cuyo efecto es, mayor satisfacción del cliente;
c. Se podrá reducir los costos de mantenimiento (parte importante del costo total del software);
d. El tiempo y costo total del ciclo de vida del software disminuirá.

Lo negativo para aplicar el SQA es:

a. Difícil de aplicar en organizaciones pequeñas, por la escasa disponibilidad de recursos para llevar a cabo las tareas;
b. Representa un cambio cultural y, el cambio nunca es fácil;
c. Requiere inversión destinada explícitamente al aseguramiento de calidad.

Caballero (2007) analizó la necesidad de buscar y adaptar modelos de procesos que permitan incrementar la calidad del software en las microempresas, la mayoría requieren una gran inversión económica y de tiempo, analizó los procesos de trabajo de microempresas de software, pertenecientes a la Sectorial de Tecnologías de la Información y Comunicación de la Unión de Profesionales y Trabajadores Autónomos de España (UPTA), para identificar y caracterizar las principales debilidades que afectan la calidad del producto software, luego construyó procedimientos en base a modelos existentes, los cuales se adaptaron al proceso base de las empresas, mostramos los resultados las tablas 2.2, 2.3 y 2.4.

Nº	Debilidades
1	Proceso ambiguo con fases y productos mal delimitados
2	Centrado en el jefe de proyecto, quien realiza los calendarios y asigna las tareas
3	Ante calendarios ajustados no se da prioridad a la calidad

4	Solamente se hacen revisiones personales, pero sin ningún control de calidad
5	Se aceptan costos y tiempos de entrega sin conocer a detalle el alcance
6	No existe un mecanismo realista para conocer los avances del proyecto
7	No existen reuniones formales frecuentes y son de carácter reactivo ante incidentes

Tabla N° 2.2: Principales debilidades identificadas en los procesos de las microempresas de desarrollo de software. (Caballero, 2007)

N°	Principio
1	Proceso bien delimitado que facilita la estimación y medición. Roles definidos
2	Filosofía de equipo: participación, colaboración y compromiso
3	Concienciación sobre la calidad con la reducción temprana de defectos
4	Introducción de actividades de inspección dentro del proceso
5	Delimitación realista y detallada del alcance para cumplir con el calendario
6	Seguimiento del proyecto a través del valor ganado
7	Reuniones semanales para analizar el avance del proyecto y resolver dudas

Tabla N° 2.3: Principios aplicados al proceso adaptado para mejorar las debilidades identificadas. (Caballero, 2007)

Objetivo	Valor Histórico	Valor Obtenido	Diferencia
Desviación de calendario	21,37%	7,70%	63,96%
Desviación de esfuerzo	55,20%	18,00%	67,39%
Desviación en costo	40,43%	15,00%	62,89%
Reducir tiempo de pruebas	24,40%	10,00%	59,10%
Reducir defectos a la entrega	14,30%	3,80%	73,40%
Incrementar la productividad en las pruebas [Horas/KLOC]	33,40%	13,20%	60,50%
Incrementar la productividad del proyecto [LOC/HORA]	7,30%	7,60%	3,90%

Tabla N° 2.4: Mejoras obtenidas con el nuevo proceso adaptado. (Caballero, 2007)

2.1.2 PROCESOS DE DESARROLLO DE SOFTWARE

Las dificultades encontradas a la hora de construir software con calidad de acuerdo a Rafael (2004) son muchas, se debe aplicar los conceptos de la ingeniería de software, para la producción del software con calidad y su mantenimiento futuro.

Existe una analogía entre RUP y ICONIX, donde el proceso ICONIX presenta modelado simple y ágil, utiliza algunos diagramas del UML como; modelo de casos de uso, diagrama de secuencia y, diagrama de clase (Rafael, 2004; ápud Ambler, 2002). El proceso ICONIX se asemeja mucho al RUP, pero sin su complejidad (Rafael, 2004; ápud Kendall, 2003) donde el proceso ICONIX se sitúa

en algún lugar entre RUP y XP, siendo relativamente pequeño y ajustado, como el XP, que mantiene el análisis y diseño durante el desarrollo.

El proceso ICONIX y el RUP están basados en el modelo iterativo e incremental, dirigidos por casos de uso, usan el UML como lenguaje de modelado, el ICONIX por ser un proceso relativamente pequeño y simple, puede ser utilizado mejor en aplicaciones más pequeñas. Para cada desarrollo de software, existe la posibilidad de aplicar diversas metodologías, los desarrolladores deben analizar la metodología que mejor se adecue a sus necesidades del desarrollo y a los usuarios. (Rafael, 2004)

La ingeniería del software, se percibe como usar la ingeniería para el desarrollo de software, caracterizada por la aplicación de principios científicos, métodos, modelos, patrones y teorías que posibilitan gerenciar, planear, modelar, proyectar, implementar, medir, analizar y, mantener un sistema de software. El desarrollo de software exige de metas medibles y, cuantificables de la calidad del software (Sommerville, 2005). En este contexto, la ingeniería de software se caracteriza por la aplicación de métodos, herramientas y procedimientos; los métodos proporcionan los detalles de cómo construir el software, las herramientas proporcionan apoyo automatizado o semi-automatizado a los métodos y, los procedimientos constituyen la unión entre los métodos y las herramientas (Pressman, 2002). Entonces, las diferentes formas de combinar estos elementos, complica la naturaleza del proyecto, que caracterizan los procesos de desarrollo de software, los cuales pueden ser procesos formales y disciplinados, como Rational Unified Process (RUP) (Kitzberger, 2005; ápud Smith, 2003). En contraparte, existen, procesos más ágiles y rápidos, como extreme programming (XP) que cubre el contexto de pequeños proyectos. (Beck, 2000)

Kitzberger (2005), considera que para realizar proyectos de mediano tamaño, surgió hace poco tiempo, un proceso que puede ser considerado de mediano tamaño (no tan grande como el RUP ni tan pequeño como el XP), el ICONIX. Este es un proceso intermedio y equilibrado, dirigido por casos de uso, como el RUP, pero no tiene la sobrecarga del RUP. Además de eso, ICONIX es relativamente pequeño y compacto, como el XP, no descarta el análisis y diseño, como lo

hace el XP. (Kitzberger, 2005; ápud Rosenberg y Scott, 2001)

Para desarrollar software de calidad, debe elegir la mejor metodología para un equipo y un determinado proyecto, tenemos dos grandes enfoques, las metodologías tradicionales y metodologías ágiles ver tabla 2.5, la primera está desarrollada para generar documentación extensa durante todo el ciclo del proyecto, la segunda pone vital importancia a los cambios rápidos durante el desarrollo, la confianza en las habilidades del equipo y mantener una buena relación con el cliente. Presentamos en las tablas 2.6 a 2.8, las diferencias, ventajas, desventajas y como poder seleccionar una metodología para un determinado proyecto de software. (Figueroa et al., 2007)

Metodologías Tradicionales	Metodologías Agiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas y normas	Proceso menos controlado, con pocos principios
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible

Tabla N° 2.5: Diferencias entre metodologías tradicionales y ágiles. (Figueroa et al., 2007)

Modelos Rigurosos	Etapa	Modelos Agiles
Planificación predictiva y "aislada"	Análisis de requerimientos	Planificación adaptativa: Entregas frecuentes + colaboración del cliente
	Planificación	
Diseño flexible y Extensible + modelos + Documentación exhaustiva	Diseño	Diseño Simple: Documentación Mínima + Focalizado en la comunicación
Desarrollo individual con Roles y responsabilidades estrictas	Codificación	Transferencia de conocimiento: Programación en pares + conocimiento colectivo
Actividades de control:	Pruebas	Liderazgo-Colaboración:

Orientado a los hitos + Gestión miniproyectos	Puesta en Producción	Empoderamiento + Autoorganización

Tabla N° 2.6: Diferencias por etapas y enfoque metodológico. (Figueroa et al., 2007)

Modelo de Proceso	Tamaño del Proceso	Tamaño del Equipo	Complejidad del Problema
RUP	Medio / Extenso	Medio / Extenso	Medio / Alto
ICONIX	Pequeño / Medio	Pequeño / Medio	Pequeño / Medio
XP	Pequeño / Medio	Pequeño	Medio / Alto
SCRUM	Pequeño / Medio	Pequeño	Medio / Alto

Tabla N° 2.7: Modelo de procesos vs. características del proyecto. (Figueroa et al., 2007)

Modelo de Proceso	Curva de Aprendizaje	Herramienta de Integración	Soporte Externo
RUP	Lenta	Alto soporte	Alto soporte
ICONIX	Rápida	Algún soporte disponible	Algún soporte disponible
XP	Rápida	No mencionado	Algún soporte disponible
SCRUM	Rápida	No mencionado	Algún soporte disponible

Tabla N° 2.8: Modelos de proceso vs. ayudas para el modelo. (Figueroa et al., 2007)

Según Ariel y Bastarrica (2005), el desarrollo de software mediante métodos ágiles con calidad de proceso y producto tiene algunas limitaciones y ventajas como: administración de requisitos viable, ninguno de los métodos define un proceso de administración de requisitos, es posible armarlo en base a artefactos (lista de requisitos, pedido, historia de usuario, rasgo), la medición y el análisis no esta definida por ninguno de los procesos ágiles, la planificación del proyecto es una constante en casi todos los métodos ágiles, la diferencia es que la planificación que se propone se aplica a iteraciones cortas y no son documentos obligatorios, son la base del trabajo, pero los planes pueden cambiar, la meta es desarrollar el producto con la satisfacción del cliente, se toma en cuenta el riesgo para priorizar las actividades planeadas, la gestión de la configuración es débil en los métodos ágiles porque máximo alcanzan elementos de gestión de configuración del código.

La práctica de producir código manualmente y mantenerlo es más crítico cuando se usa un modelo de proceso iterativo e incremental, usar patrones de diseño asocia un código extensamente utilizado y probado con propiedades de calidad. Disponer de un entorno de desarrollo que combine patrones de diseño y refactoring permitirá desarrollar nuevo código de forma rápida, con confianza y mejorar la calidad del software, el código existente requerirá

menos esfuerzo en refactoring por tratarse de código generado a partir de patrones, la automatización de ambas tareas agilizaría el desarrollo, una herramienta capaz de detectar una situación de refactoring o patrones de diseño ayudaría mucho el trabajo del programador. (Sánchez et al., 2000)

No existe una metodología universal, para tener éxito con cualquier proyecto de desarrollo de software, se debe adaptar una al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.). Las metodologías ágiles ofrecen una solución a una gran cantidad de proyectos pequeños y medianos, una cualidad destacable es su sencillez, para su aprendizaje y aplicación, reduciendo los costos de implantación del equipo de desarrollo. Sin embargo, existen una serie de inconvenientes y restricciones para su aplicación, como: estar dirigido a equipos pequeños o medianos, usar tecnologías que no tienen un ciclo rápido de realimentación o que no soportan fácilmente el cambio, etc., escasez de estudios sobre aspectos teóricos y prácticos de su uso. La actividad de investigación en metodologías ágiles, está orientada hacia líneas como: métricas y evaluación del proceso, herramientas específicas para apoyar las prácticas ágiles, aspectos humanos y de trabajo en equipo. (Letelier y Penadés, 2006)

2.1.3 LA METODOLOGIA PROGRAMACION EXTREMA (XP)

La programación extrema XP (Extreme Programming) es un método atribuido a Kent Beck, Ron Jeffries y Ward Cunningham (Beck, 1999). El objetivo de XP esta dirigido a grupos pequeños y medianos de construcción de software, donde los requisitos aún son muy vagos, cambian rápidamente o son de alto riesgo. Las recomendaciones de XP están encaminadas a producir software de calidad. XP fue diseñado teniendo en cuenta los problemas que existían con las metodologías tradicionales de programación en cuanto a tiempos de entrega y satisfacción del cliente. El objetivo principal para XP es buscar la satisfacción del cliente tratando de mantener durante todo el tiempo su confianza en el producto. (Ariel y Bastarrica, 2005)

XP provee elementos básicos para administrar los requisitos, el cliente en el proyecto es un actor clave para la administración de los requisitos, en XP se rastrea las estimaciones hechas por el equipo como esfuerzo y tiempo, la

planificación permite contestar dos preguntas clave durante el desarrollo de software: predecir lo que se hará ahora, y determinar que se hará después. La historia de usuario es utilizada como unidad de medida tanto para la gestión, como para la calidad, el cliente es un actor activo en la planeación, donde el equipo técnico estima el esfuerzo requerido para implementar las historias de usuario y los clientes deciden sobre el ámbito y el tiempo de las entregas. En XP el código es el principal artefacto y el aseguramiento de la calidad se centra en el código con la programación por pares, la refactorización y la prueba, el aseguramiento de la calidad de las historias de usuario se logra con la participación activa del cliente. (Ariel y Bastarrica, 2005)

La comunidad de XP ha crecido considerablemente, era tiempo de hacer una encuesta sobre XP, indicamos los resultados sobre 45 preguntas realizadas el 2001. Se identificó como un riesgo alto la falta frecuente del cliente, los problemas están relacionados a la "barrera mental del directivo", porque no se admite a los clientes, los programadores son negados a trabajar en pareja. Los elementos útiles de XP son: la propiedad común del código, la prueba de integración de forma continua, la programación en pareja y el enfoque de XP sobre los objetivos del cliente. En conclusión, a la pregunta si usaría otra vez XP, el 93.3 % responde que si y 6.7 % requiere que se mejore. Creemos que XP debe pertenecer al dominio de un ingeniero de software, también otras técnicas tradicionales, que permita al ingeniero reaccionar de forma flexible frente a diversos proyectos. (Rumpe y Schröder, 2001)

Existen problemas frecuentes al desarrollar software, que son: llega la fecha de entregar el software pero no esta disponible, el software creado luego de un par de años tiene costo alto de mantenimiento, el software esta en producción pero presenta muchos defectos, el software no resuelve los requisitos planificados, el problema que resolvía nuestro software ha cambiado y nuestro software no se adapta. XP trata de evitar estos riesgos, pero tiene críticas contra la programación por pares, el mito de las 40 horas semanales, que solo funciona con programadores muy buenos como Kent Beck que son capaces de hacer un buen diseño. XP es mas una filosofía de trabajo que una metodología, suponemos que: las personas son clave en los procesos de

desarrollo, los programadores son profesionales que no necesitan supervisión, los procesos se aceptan y se acuerdan, desarrolladores y gerentes comparten el liderazgo del proyecto, el trabajo de los desarrolladores con los usuarios debe ser regular, no esporádico. (Rumpe, 2001)

Las prácticas de desarrollo de software con XP obtienen el máximo rendimiento con equipos pequeños de programadores, mínimo soporte de procedimientos y documentación del proceso (Sicilia et al., 2002), proponemos completar algunas prácticas de XP con procedimientos "ligeros" de control y documentación, denominados micro-prácticas, a fin de aumentar el grado de confiabilidad del proceso XP, pero asociadas a las tareas esenciales de XP para hacerlas aceptables dentro de su filosofía, como mostramos en tabla 2.9. P "es programador" y M "es manager".

Práctica	**XP Micro-práctica**	**Resp.**	**Beneficios desde el Punto de Vista del Control**
Rediseño (*refactoring*)	Documentar el propósito del rediseño	P	Es una técnica informal que sustituye aprobaciones del diseño formales (requeridas en el objetivo "aprobación del diseño").
	Auditoria técnica de los rediseños	P	Alineamiento parcial con los objetivos de identificación y medición de riesgos y el plan de acción contra riesgos. También, sirven como base para la definición de métricas de calidad del rediseño como se requiere en métricas de calidad.
Análisis de requisitos mediante historias de usuario (*user stories*)	Matriz de traza de historias de usuario	M, P	Documento esencial para la trazabilidad y auditoria del desarrollo, y puede cubrir parcialmente los requisitos del objetivo "especificaciones de programas".
	Estructuración de las historias de usuario	M	El objetivo "definición de requisitos de información" necesita una especificación de los requisitos previo al inicio del desarrollo, que podría cubrirse con esta documentación concisa.
Planificación por iteraciones	Auditoria de cambios en la planificación	M	Alineamiento parcial con los objetivos de identificación y medición de riesgos y el de plan de acción contra riesgos.
Diseño evolutivo	Esquemas de arquitectura	P	Incorpora actividades mínimas para cumplir con el objetivo "métodos de diseño". También, proporciona la posibilidad de incorporar análisis de viabilidad tecnológica mediante el prototipado, cumpliendo parcialmente el objetivo "estudio de factibilidad tecnológica".

Tabla N° 2.9: Resumen de extensiones propuestas. (Sicilia et al., 2002)

Los antecedentes de XP, están en los trabajos de Ward Cunningham quien propone un desarrollo de software simple y eficiente, en 1989, Cunningham formó un equipo que usaba los principios y muchas de las prácticas que después adoptaría XP, mientras trabajaba para la empresa "Wyatt Software" (González y Fernández, 2006; ápud Fowler, 2000). Un aspecto de XP es que representa la antítesis del proceso tradicional de desarrollar software, XP es una metodología "liviana" que obvia los casos de uso, la exhaustiva definición de requerimientos y la producción de una documentación extensa (González y Fernández, 2006; ápud English, 2002), XP tiene asociado un ciclo de vida y se considera un proceso (González y Fernández, 2006; ápud Amber, 2002). Existen aspectos positivos de XP que son: pruebas unitarias del código como práctica y factor clave para obtener software de alta calidad, integración continua para evitar problemas por defectos no detectados a tiempo, énfasis en la simplicidad y el refactoring, 40 horas por semana para el programador, alto valor a la comunicación, rol protagónico del programador en planeación, cliente con presencia constante junto al desarrollador, "escuchar al cliente" una de las cuatro actividades esenciales de la programación. También, existen aspectos controversiales como: equipos de trabajo no más de 20 programadores, pesando para usar tecnologías que permiten pruebas e integración continúa, XP desalienta el diseño, débil en la documentación, modelo que no pasa para proyectos con seguridad crítica, exceso de pruebas que retrasa el desarrollo, diseño simple solo pasa para proyectos simples, la programacion en pares consume mayor tiempo y recursos (González y Fernández, 2006; ápud Aiken, 2004), XP asume implícitamente la programación orientada a objetos, el refactoring se ve como sinónimo de rediseño constante, la planeación no debería hacerse "sobre la marcha" por la experiencia en proyectos fracasados (González y Fernández, 2006; ápud Glass, 2001). Hasta ahora no existen estudios cuantitativos que midan el grado de éxito obtenido al adoptar este paradigma. (González y Fernández, 2006)

Los diez métodos ágiles más usados son; extreme programming, scrum, evo, crystal methods, feature driven development, RUP, dynamic systems development method, adaptive software development, agile modeling y lean development, las encuestas sobre los métodos ágiles son demasiado

complejas y heterogéneas, la programación extrema de acuerdo a encuestas de Cutre Consortium tiene el 38% del mercado, FDD el 23%, Adaptive Software Development el 22%, DSDM el 19%, Crystal el 8%, Lean Development el 7% , Scrum el 3% y, los demás el 9%. (Reynoso, 2004)

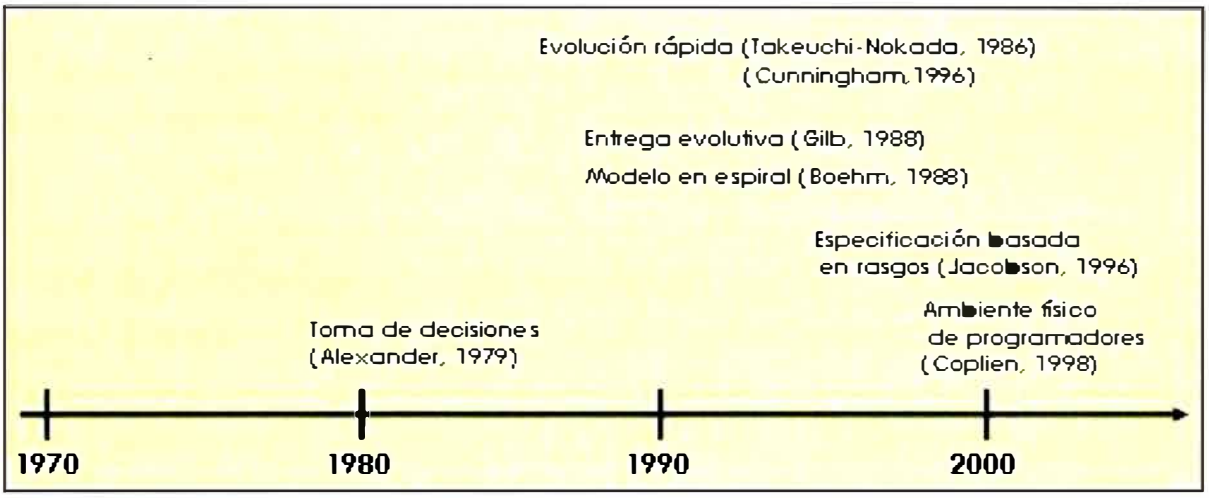

Figura N° 2.2: Gestación de la Programación Extrema. (Reynoso, 2004).

Según Reynoso (2004), los obstáculos más comunes en los proyectos XP, es pretender que el cliente trabaje con el desarrollador y la resistencia de muchos programadores a trabajar en pares. Craig Larman señala como factor negativo la ausencia de arquitectura durante las primeras iteraciones y la consiguiente falta de métodos de diseño arquitectónico. XP ha sido, de todos los métodos ágiles, el que más resistencia ha encontrado, en una entrevista a Kent Beck y Martin Fowler, Beck afirmó que mientras SEI se encuentra en el campo modernista, XP es más bien posmoderno, y "Sus raíces filosóficas se encuentran en la teoría de los sistemas complejos" y sus capacidades se generan a través de procesos emergentes. (Beck, 1999)

Las 12 reglas de XP son excesivamente rígidas, casi nadie las aplica en su totalidad, Tom Gilb proponía lineamientos similares muchos años atrás. En diversas presentaciones públicas McConnell ha considerado la programación sin diseño previo, como la programación automática y la costumbre de llamar "ágil" a cualquier práctica como algunas de las peores ideas en construcción de software del año 2000. Un miembro de Rational, John Smith opina que la terminología de XP esconde una complejidad no reconocida, mientras las

palabras "artefacto" y "producto de trabajo" no figuran en los índices de sus libros canónicos, Smith cuenta más de 30 artefactos encubiertos como: historias, tareas, tareas técnicas, pruebas de aceptación, código de software, entregas, metáforas, diseños, documentos de diseño, estándares de codificación, unidades de prueba, plan de entrega, plan de iteración, reportes y notas, plan general y presupuesto, reportes de progreso, estimaciones de historias, estimaciones de tareas, defectos, datos de prueba, herramientas de prueba, herramientas de gestión de código, resultados de pruebas, spikes, registros de tiempo de trabajo, datos métricas, resultados de seguimiento. Según Smith, la lista es indicativa, no exhaustiva, porque para un proyecto pequeño, RUP demanda menos que eso, al no tratar sus artefactos como tales. Asimismo, en toda la industria se sabe que la refactorización en general no esta muy bien, se manifiesta en proyectos grandes con requisitos no funcionales clásicos "quebradoras de arquitectura", como procedimientos especiales para resolver problemas de eficiencia, seguridad o tolerancia a fallas, aquí como dice Barry Boehm, ninguna dosis de refactorización será capaz de armar a Humpty Dumpty. (Reynoso, 2004)

2.1.4 LA METODOLOGIA ICONIX

ICONIX esta considerado como una metodología pura, práctica y simple, con el componente de análisis y representación del problema sólido y eficaz, ICONIX es un proceso no tan burocrático como el RUP, no genera tanta documentación, es un proceso simple como XP, no deja de hacer el análisis y diseño, se destaca como un poderoso proceso de análisis de software, este proceso hace uso del lenguaje de modelado (UML) y tiene una característica exclusiva llamada "rastreabilidad de los requisitos". ICONIX es flexible y abierto, es decir, si es necesario usar otro recurso del UML para complementar los artefactos en las fases del ICONIX se puede incluir. ICONIX esta compuesto por los artefactos: modelo de dominio, modelo de caso de uso, diagrama de robustez, diagrama de secuencia y diagrama de clase. (Pantoja, 2009)

Según Pantoja (2009) el proceso ICONIX esta dividido en dos grandes partes, desarrollados en paralelo y de forma recursiva, siendo el modelo estático y dinámico, el modelo estático esta formado por los diagramas de dominio y de

clases que modela el funcionamiento del sistema, el modelo dinámico siempre muestra al usuario interactuando con el sistema presentando una respuesta a una acción del usuario. El modelo estático es refinado progresivamente durante las iteraciones del modelo dinámico, ICONIX comienza con un prototipo de interface, luego se desarrollan los casos de uso basado en los requisitos del usuario, a continuación se hace el análisis de robustez para cada caso de uso, posteriormente se desarrolla el diagrama de secuencia, se actualiza el modelo de dominio y de clases, luego se implementa.

ICONIX fue creado por Rosenberg y Scott en la década de 1990, caracterizado por ser un proceso de desarrollo simplificado, usa la orientación a objetos y considera las fases de análisis, diseño e implementación del ciclo de vida del desarrollo de software, su esencia puede resumirse como "definición de modelos de objetos a partir de los casos de uso". Los recursos utilizados por ICONIX son suficientes para representar el trabajo de modelado, esta dirigido por casos de uso, presenta proceso iterativo e incremental, mantiene enfoque direccionado a la rastreabilidad de los requisitos. ICONIX proporciona respuestas a preguntas básicas del software, algunas preguntas (De Matos, 2005; ápud Silva y Videira, 2001) son:

a. ¿Quiénes son los usuarios y que hacen?. Son actores del sistema, usan los casos de uso;
b. ¿Qué son en el "mundo real"?. dominio de problema, ¿Qué objetos y asociaciones existen entre ellos?. Diagramas de clases de alto nivel;
c. ¿Qué objetos son necesarios para cada caso de uso?. Diagramas de robustez;
d. ¿Cómo los objetos están colaborando e interactuando en cada caso de uso?. Diagramas de secuencia;
e. ¿Cómo será construido el sistema a nivel práctico?. Diagramas de clases de bajo nivel.

Para Soares (2009), desarrollar software esta compuesto por planear, proyectar y evaluar el software a ser construido, son fases cuidadosamente planeadas para proveer productos con calidad, bajo costo y rapidez en la construcción. En este contexto, ICONIX es una metodología de desarrollo de software

promovida por la empresa *ICONIX Software Engineering*, dirigido por casos de uso, es iterativo e incremental, es relativamente pequeño y simple, como XP, pero sin eliminar las tareas de análisis y diseño que XP no contempla. El corazón del proceso ICONIX reside en la filosofía, que para construir buenos modelos de objeto hay que responder preguntas importantes acerca del sistema que esta construyendo y rechazar a involucrarse en asuntos superfluos de modelado. Según Scott (2001), "la diferencia entre la teoría y la práctica es que en teoría, no hay diferencia entre teoría y práctica, pero en la práctica, si hay".

El estado del arte sobre el proceso ICONIX, sería:

a. Es un proceso práctico, modela un sistema de software según la orientación a objetos;
b. Es una metodología práctica e intermedia, esta entre la complejidad del RUP y la simplicidad del XP, sin dejar de hacer análisis y diseño;
c. Esta conducido por casos de uso, es iterativo e incremental;
d. Distingue con claridad los requisitos y casos de uso;
e. Presenta alto grado de rastreabilidad de requisitos, verificando que en las fases del ciclo de vida, los requisitos están siendo atendidos;
f. No usa varios diagramas UML como, diagramas de estado, de actividad, de arquitectura y de colaboración;
g. La idea sobre el proceso ICONIX seria "hacer lo menos posible y, construir en el menor tiempo, para tener un software de calidad".

2.1.5 COMERCIALIZACION DE LA TARA EN LA REGION AYACUCHO

A nivel internacional la tara Ayacuchana no tiene problemas de calidad, porque la concentración de taninos, satisface plenamente las exigencias del mercado internacional, la vaina de tara es un producto obtenido sin desarrollado de técnicas de cosecha y post cosecha orientadas a mejorar la calidad y cantidad, que se comercializa sin ningún tipo de selección, la informalidad en las relaciones entre los actores de la cadena es una constante, salvo entre acopiadores mayoristas y transformadores que vienen implementado algún tipo de formalidad en sus relaciones comerciales. Existe limitada información para los actores de la cadena respecto al

producto, a las condiciones del mercado, generando expectativa en el precio. Se recomienda que las entidades del Gobierno y las ONGs deban dar énfasis al trabajo cooperativo de productores y acopiadores, para aprovechar al máximo la aceptación que tiene la tara en el mercado, facilitar información a los productores y acopiadores locales respecto al producto, a las condiciones de oferta y demanda para posicionarlo en el mercado internacional. (Pillpe y Quispe, 2007)

La Región Ayacucho, cuenta con suficiente materia prima (7039 TM), la tara es un recurso forestal de gran importancia debido a sus múltiples usos y productos que se pueden obtener al ser aprovechados integralmente, en la vaina de la tara la pepa constituye el 38%, el polvo mas fibra es 62%, de la vaina es posible obtener polvo, goma, alimento balanceado y bio abono, siendo los dos primeros de mayor importancia en el mercado exterior, el análisis de mercado presenta una tendencia de crecimiento al consumo de productos naturales como goma y polvo de tara, donde los mercados principales para la goma son; Alemania, Estados Unidos, Corea, Japón, y España, para el polvo son; Bélgica, Italia, Estados Unidos y China. En el Perú existen diferentes empresas procesadoras – exportadoras, quienes prefieren acopiar materia prima de la Región Ayacucho debido a su mejor calidad, principalmente por el alto contenido de taninos (56%) y menor porcentaje de humedad. (Bautista y Bendezú, 1998)

El año 1993 se realizó el "Estudio de Caracterización del Proceso Económico de Tara" en los Departamentos de Ayacucho, Huancavelica y Apurímac, primer conocimiento de la cadena, identificando 15,528 productores de tara, ubicados en 559 localidades. (Avendaño et al., 2007; ápud Palomino et al., 2001)

El año 2004, el Programa de Desarrollo Rural Sostenible (PDRS – GTZ), realizó el primer estudio de mercado de tara, identificando el crecimiento de la demanda de polvo y goma de tara, describiendo los productos sustitutos en el mercado de taninos y se plantearon sugerencias para mejorar los aspectos productivos. (Avendaño et al., 2007; ápud Schiaffino, 2004)

Por iniciativa de IDESI Ayacucho y SNV, se realizó el análisis participativo de la cadena productiva de tara en Ayacucho, identificando que hasta el año 2003 había una producción de 3,500 TM de tara en vaina, donde el 91% procedía de las provincias Huanta y Huamanga, se identificó alta informalidad en la comercialización, estableciéndose que la tasa aproximada de crecimiento de la demanda mundial de goma de tara era cercana al 20% anual. (Avendaño et al., 2006)

En Ayacucho se encuentra tara en plantaciones y silvestre, el año 2006 se estima que existen 810 Ha con rendimiento promedio de 24.44 Kg vaina/árbol, 15,000 productores y 5,083 TM de producción de tara en vaina (Huanta, 67%; Huamanga, 26%; Cangallo 2%; La Mar, 2% y otras provincias, 3%). La oferta productiva es limitada frente a la demanda mundial, siendo comercializada toda la producción, el 74% se comercializa en vaina y el 26% transformado en goma y harina, los principales demandantes del mercado Ayacuchano son: Silvateam Perú 35%, Productos del País 30%, Exportadora el Sol 8% y otros 27%. Las principales empresas exportadoras son: Silvateam Perú 33.7%, Exportadora el Sol 17.7%, Transformadora Agrícola 10.7%, Productos del País 9%, Molinos Chipoco 9%, Exportaciones de la Selva 7%, otras empresas 12%. Los principales países de destino de harina de tara son: Italia 22%, Brasil 17%, Argentina 16%, China 7%, otros países 38%. En goma de tara: Estados Unidos 14%, España 13%, Alemania 11%, Italia 10%, Argentina 8%, Japón 6%, otros países 38%, el precio promedio de exportación para la harina de tara y el preparado curtiente es casi estable, para la goma de tara subió de US$ 2.86/ Kg el año 2004 a US$ 5.34/ Kg a junio del 2007. Las relaciones entre actores directos e indirectos esta planteado en la tabla 2.10. Las relaciones comerciales del mercado local predomina la informalidad, que genera desconfianza, falta de transparencia y deslealtad en las transacciones comerciales. La Región Ayacucho ha generado US$ 4´765,155 por comercializar tara, 1,238 empleos, con 56.4% en producción. Debemos resolver: la limitada capacidad de productores y empresarios (conocimiento, nivel educativo, tecnológico, recursos económicos, etc.), alta informalidad en comercialización, y escasa cultura de asociatividad. (Avendaño et al., 2007)

Tipo	Eslabones			
	Producción	Acopio	Transformación	Consumo
Actor Directo	Recolectores. Productores. Productores empresarios.	Acopiadores pequeños. Acopiadores medianos. Acopiadores mayoristas.	Transformadores-exportadores grandes, mediano y pequeños	Industria curtiembre. Industria farmacéutica. Industria alimenticia.
Actor Indirecto	INRENA, IDESI Ayacucho, Vecinos Perú, DRA Ayacucho, FONCODES, FCA - UNSCH, PRONAMACHCS, CEDESUR, Municipios Provinciales	Transportistas de carga, estibadores, entidades financieras, INRENA, IDESI Ayacucho, Vecinos Perú, FONCODES	Operarios para el procesamiento, servicios de mantenimiento de planta, entidades financieras, INRENA, transportista de Carga, SUNAT - ADUANAS, ADEX, PROMPEX	Entidades financieras. Navieras. Aduanas.

Tabla N° 2.10: Actores directos e indirectos de la cadena tara. (Avendaño et al., 2007)

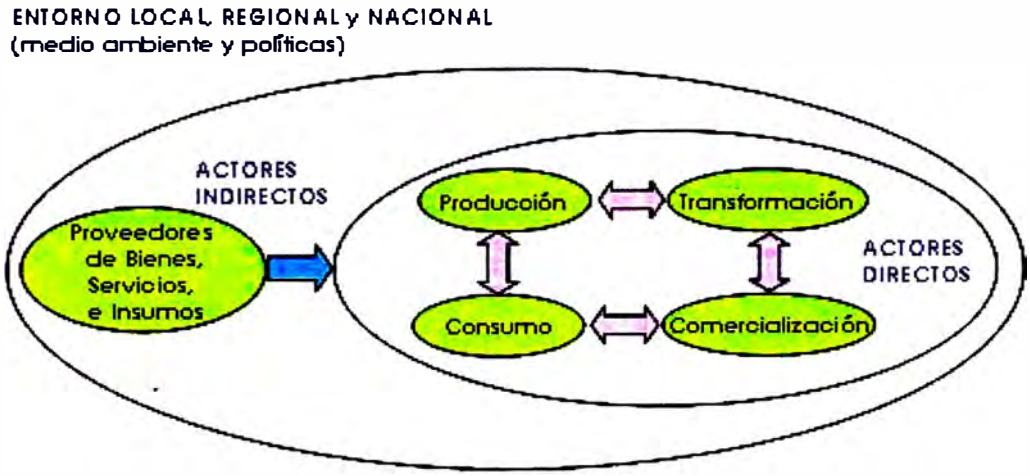

Figura N° 2.3: Modelo simplificado de cadena productiva.

2.2 MARCO TEORICO

Se realiza la revisión literaria sobre los enfoques teóricos y conceptuales, y teorías que existen sobre las variables de investigación. Asimismo, los métodos y técnicas existentes para abordar el problema.

Ingeniería del Software

Existe una infinidad de definiciones de ingeniería del software, citaremos las más importantes:

"La ingeniería del software es el establecimiento y uso de principios robustos de

la ingeniería a fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales". (Pressman, 2002:14; ápud Bauer, 1969)

"Ingeniería del software: (1) La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software. (2) El estudio de enfoques como en (1)". (Pressman, 2002:14; ápud IEEE, 1993)

"La ingeniería del software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza. En esta definición, existen dos frases clave: disciplina de la ingeniería y todos los aspectos de producción de software". (Sommerville, 2005:6)

"La ingeniería de software comprende un proceso, métodos técnicos y de gestión, y herramientas". (Pressman, 2002:14)

Software

"El software se compone de programas, datos y documentos. Cada uno de estos elementos compone una configuración que se crea como parte del proceso de la ingeniería del software". (Pressman, 2002:10)

Software son programas de computadora, la documentación asociada y la configuración de datos que se necesitan para hacer que estos programas operen de manera correcta. Los productos de software se pueden desarrollar para algún cliente o para un mercado general. (Sommerville, 2005:5)

"Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación". (ANSI/IEEE, 1983)

2.2.1 PROCESO DEL SOFTWARE

Existen términos usados de forma indistinta en la literatura de la

ingeniería del software, estos son; proceso, procedimiento, actividad, tarea, fase, ciclo de vida, modelo, método y metodología. Para tener definiciones coherentes las desarrollamos a continuación.

"El proceso es un conjunto de actividades cuya meta es el desarrollo o evolución del software". (Sommerville, 2005)

"…definimos un proceso de software como un marco de trabajo de las tareas que se requieren para construir software de alta calidad. (Pressman, 2002:13)

"El proceso proporciona una interacción entre los usuarios y los diseñadores, entre los usuarios y las herramientas de desarrollo y, entre los diseñadores y las herramientas de desarrollo (tecnología). Es un proceso interactivo donde la herramienta de desarrollo se usa como medio de comunicación, con cada interacción se obtiene mayor conocimiento de las personas involucradas". (Pressman, 2002:13; ápud Baetjer, 1998) que indican lo siguiente:

a. El proceso esta compuesto por actividades, si agrupamos actividades, podemos usar el término fase para describir actividades de más alto nivel;
b. Las definiciones implícitamente contemplan procedimientos y métodos, herramientas y equipos, comunicación y personas;

Los procesos para producir software, proporcionan mayor importancia a la calidad del producto software y la productividad del equipo, no existe un modelo único que describa con precisión lo que ocurre durante todas las fases de producción de un software, porque los procesos y las necesidades de cada usuario son diferentes.

A. Fases del Proceso de Software

Una definición inicial de las fases que tiene un proceso de software, de acuerdo a Schwartz (1975), es:

a. Especificación de requisitos.- necesidad o requisito operacional para una descripción de la funcionalidad a ser ejecutada;
b. Diseño de sistema.- interpretación de estos requisitos en una descripción de todos los componentes necesarios para codificar el sistema;

c. Programación.- producción del código que controla el sistema y realiza la computación y lógica involucrada;
d. Verificación e integración (checkout).- verificación de la satisfacción de los requisitos en el producto producido.

Otra definición actual proporcionada por Sommerville (2005) es similar, define las fases como: especificación, desarrollo, validación y evolución que se organizan de forma distinta para diferentes procesos de desarrollo, la ultima fase no esta descrita directamente por Schwartz (1975), donde la evolución (mantenimiento) es el cambio del software para tener en cuenta las nuevas necesidades del usuario. Pressman (2002:15), proporciona una visión simplificada donde, el software se crea aplicando tres fases distintas que son; definición, desarrollo y mantenimiento.

Respecto a la aparente secuencialidad de las fases, diversos autores opinan que el proceso de software es iterativo y cíclico que la idea de ser simples fases. Rosenberg et al. (2005), Beck (1999) y Schach (2005), sugieren procesos con ciclos continuos e iterativos, donde un producto es desarrollado en cada ciclo. Sobre la extensión de cada ciclo y el procedimiento no existe consenso.

B. Ciclo de Vida del Software

Es una sucesión de fases por las que atraviesa un producto software a lo largo de su desarrollo y existencia. Cada fase esta compuesto por un conjunto de actividades que son ejecutadas. De acuerdo a las clasificaciones de Schwartz (1975), Pressman (2002) y Sommerville (2005), podemos aproximar las siguientes fases: especificación, diseño, implementación, validación, mantenimiento y evolución. Existen subproductos que son generados en cada fase, ejemplo en la fase de especificación, se ha desarrollado y entregado uno o más documentos que detallan los requisitos del sistema, estos subproductos son denominados entregables o artefactos.

C. Modelos del Ciclo de Vida del Software

Es una representación de las fases del proceso de software y sus interdependencias. Según Sommerville (2005), los modelos de ciclo de vida

representan un marco de trabajo pero no proporcionan detalles de las actividades especificas, los modelos nos permiten explicar diferentes enfoques del desarrollo de software. A continuación describimos el modelo iterativo e incremental de acuerdo al marco de la investigación.

Modelo Iterativo e Incremental

Larman (2000:20) define que "un ciclo de vida iterativo se basa en el agradamiento y perfeccionamiento secuencial de un sistema a través de múltiples ciclos de desarrollo de análisis, diseño, implementación y pruebas", como se observa en la figura 2.4.

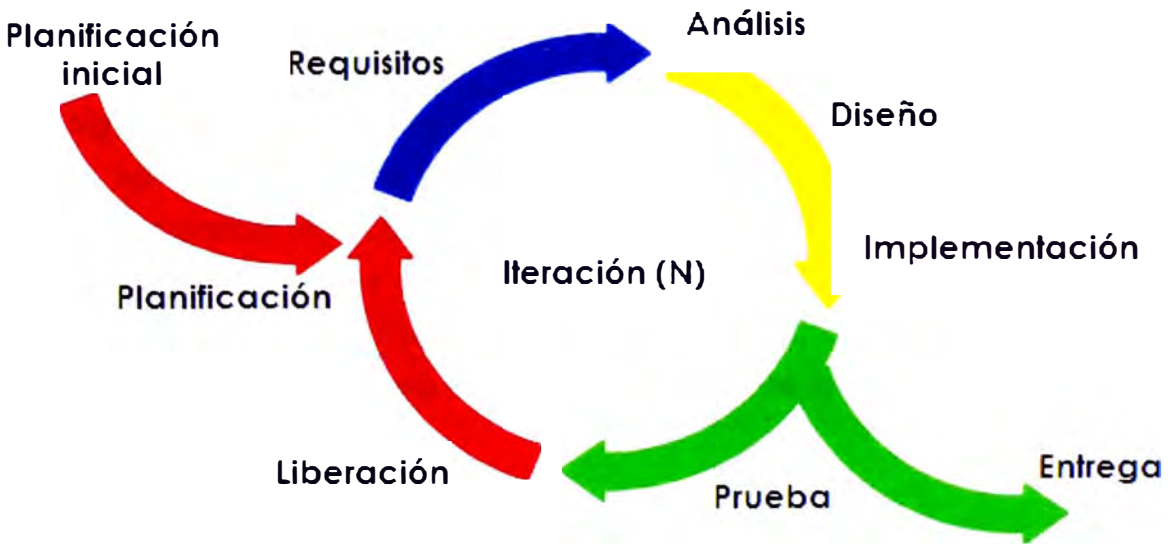

Figura N° 2.4: Modelo Iterativo

Jacobson et al. (2000), presenta las actividades del modelo iterativo e incremental para el proceso unificado en cuatro fases: inicio, elaboración, construcción y transición, fases que están organizadas en actividades. Al término de cada fase se logra un hito, pero no se termina la ejecución completa de las actividades. Cada fase tiene como propósito lo siguiente:

a. Inicio.- comprensión inicial y definición del producto, es decir, lo que se entregue;
b. Elaboración.- comprensión inicial y concordancia del proyecto detallado, es decir, como será construido;
c. Construcción.- creación de la primera construcción totalmente funcional;
d. Transición.- entrega del producto de acuerdo con los requisitos iniciales.

Existen beneficios que Jacobson et al. (2000) indican por adoptar un modelo iterativo e incremental, podemos destacar los siguientes:

a. Reducir el riesgo por invertir solo para un incremento, si el equipo necesita repetir la iteración, se pierde únicamente el costo de la iteración;
b. Reducir el riesgo por entregar el producto fuera de la fecha prevista, por identificarlo al inicio y no la final del proyecto;
c. Acelerar el tiempo de desarrollo del proyecto completo, porque los desarrolladores trabajan de forma más eficiente cuando tiene metas más pequeñas y claras;
d. Reconocer una realidad ignorada, las necesidades de los usuarios (requisitos) no pueden ser totalmente definidos al inicio del proceso, éstos son refinados en iteraciones sucesivas.

En las siguientes secciones presentaremos el proceso ágil y formal ICONIX y el proceso ágil XP. El método (hacer con orden) en ingeniería de software nos dice como construir técnicamente el software, la metodología es el tratado de los métodos. Entonces, la metodología es el tratado de los métodos de la ingeniería de software, en términos simples, decimos, que la metodología es una receta que tenemos para construir técnicamente el software.

2.2.2 EL PROCESO ICONIX

ICONIX es un proceso simplificado que unifica un conjunto de métodos orientados a objetos, para dar cobertura al ciclo de vida del software, elaborado por Doug Rosenberg y Kendall Scott a partir de la síntesis del proceso unificado de Jacobson et al. (2000), que proporciona soporte y comprensión a la metodología ICONIX.

En la figura 2.5, observamos los principios de la concepción del ICONIX, donde: OMT de Rumbaugh aporta con su modelo de objetos de dominio del problema, Objetory de Jacobson con el modelo de dominio de solución dirigida por el usuario y Booch con sus modelos a nivel de diseño detallado.

La metodología consiste en producir un conjunto de artefactos con parte dinámica y estática del sistema, desarrollados incrementalmente y en paralelo,

el modelo estático se incrementa y es refinado por el modelo dinámico.

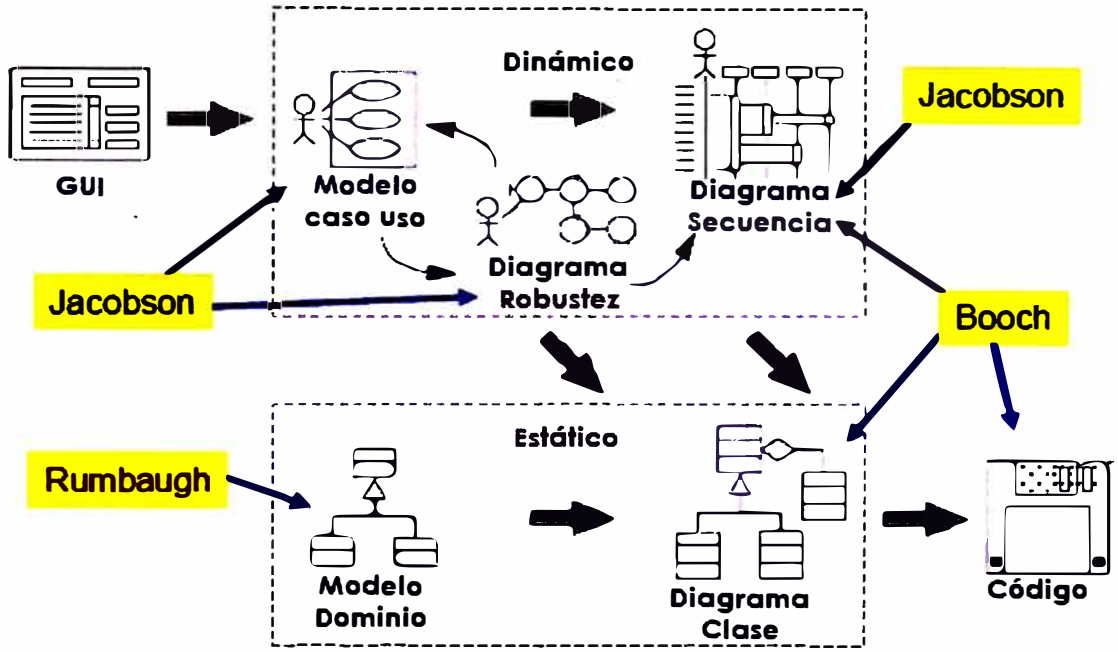

Figura N° 2.5: Esquema del proceso ICONIX, adaptado. (Rosenberg y Scott, 2001)

Características fundamentales del proceso ICONIX:

a. El proceso es conducido por casos de uso;
b. Relativamente pequeño y simple, tal como el XP, pero sin eliminar el análisis y diseño que XP no contempla;
c. Uso simplificado del UML (OMG®, 2001) mediante: modelo de dominio, diagrama de casos de uso, diagrama de robustez, diagrama de secuencia y diagrama de clases;
d. Alto grado de trazabilidad, para seguir la relación entre los artefactos producidos y los requisitos.

Rosenberg y Scott (1999), Rosenberg y Scott (2001), Rosenberg et al. (2005) y Rosenberg y Stephens (2007), presentan las fases: análisis de requisitos, diseño preliminar, diseño detallado, implementación y pruebas, con sus hitos, que detallados a continuación.

A. Análisis de Requisitos

a. Definir lo que el sistema debe ser capaz de hacer identificando los requisitos funcionales y no funcionales;
b. Identificar los objetos del "mundo real" y sus relaciones de agregación y

generalización, presentarlo en el diagrama de clase de alto nivel denominado modelo de dominio;

c. Realizar un prototipo rápido de interfase hombre - maquina (GUI), para que el cliente pueda comprender mejor el sistema en desarrollo;

d. Identificar los casos de uso del sistema, mostrando los actores involucrados, presentarlo usando el diagrama de casos de uso;

e. Organizar los casos de uso lógicamente en grupos, presentarlo utilizando los diagramas de paquetes;

f. Asignar los requisitos funcionales a los casos de uso y objetos de dominio;

g. Escribir el primer borrador de casos de uso.

Primer Hito: Revisión de los requisitos.- La descripción de los casos de uso debe coincidir con las expectativas del cliente, revisar los casos de uso por lotes pequeños, antes de diseñarlos.

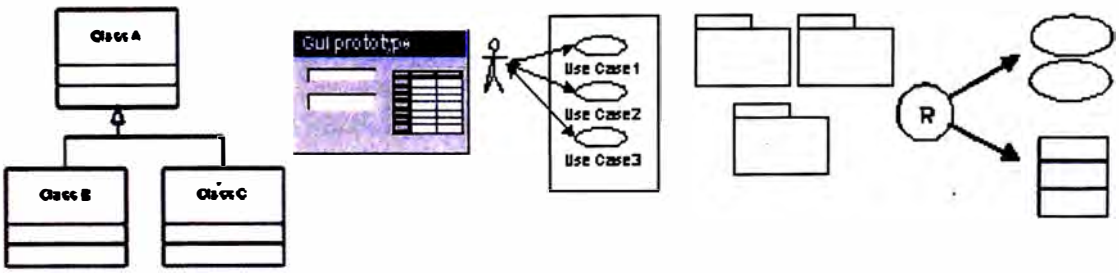

Figura N° 2.6: ICONIX – Tareas del análisis de requisitos. (Rosenberg y Scott, 2001)

ICONIX distingue explícitamente un requisito de un caso de uso. Un caso de uso describe un comportamiento del sistema y un requisito describe una regla para el comportamiento. Además, un caso de uso satisface uno o más requisitos funcionales, un requisito funcional puede ser satisfecho por uno o más casos de uso. Aún estando de acuerdo con los autores, la relación entre casos de uso y requisitos esta en discusión en la comunidad de orientación a objetos, no existiendo consenso hasta el momento. (Rosenberg, y Stephens, 2007)

Importante, en cada iteración, es decir, para un pequeño grupo de casos de uso, hacer diseño preliminar, diseño detallado, implementación y pruebas.

B. Diseño Preliminar

a. Rescribir (desambiguar) los casos de uso, usando la plantilla, curso normal y curso alterno de la interacción hombre - maquina;

b. Hacer el análisis de robustez, para cada caso de uso, identificar un conjunto de objetos (interface, control y entidad), actualizar el modelo de dominio con nuevos objetos y atributos descubiertos;

c. Finalizar actualizando el diagrama de clase a nivel de análisis.

Segundo Hito: Revisión del diseño preliminar.- Asegurarse que los diagramas de robustez, el modelo de dominio y, los casos de uso concuerdan mutuamente. Hacer la revisión para cada paquete de casos de uso.

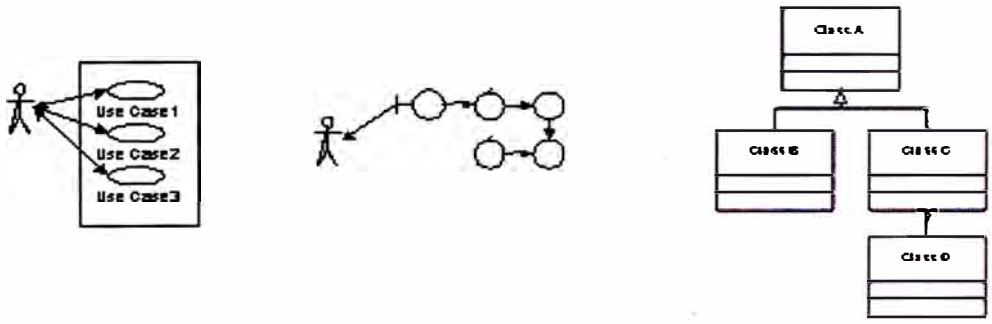

Figura N° 2.7: ICONIX - Tareas del diseño preliminar. (Rosenberg y Scott, 2001)

C. Diseño

a. Dividir el modelo de dominio actualizado para cada caso de uso;

b. Dibujar un diagrama de secuencia para cada caso de uso;

c. Actualizar el diagrama de clases para un caso de uso;

d. Generar pruebas unitarias.

Tercer Hito: Revisión crítica de diseño.- Para el grupo actual de casos de uso, persiga tres objetivos: asegúrese que el diseño detallado concuerda con la especificación de los requisitos, revisar la calidad del diseño, verificar la continuidad de mensajes en los diagramas de secuencia.

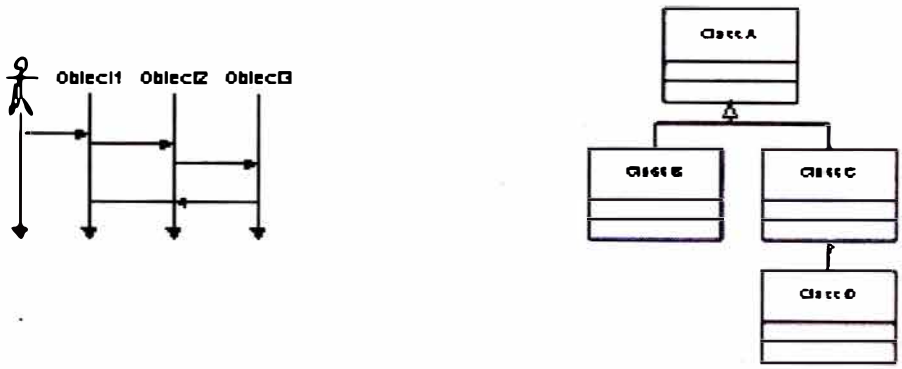

Figura N° 2.8: ICONIX - Tareas del diseño. (Rosenberg y Scott, 2001)

D. Implementación

a. Si necesita construya los diagramas de componente y despliegue, para apoyar la implementación;

b. Escribir/generar el código;

c. Ejecutar las pruebas unitarias y de integración;

d. Ejecutar pruebas de sistema y aceptación del usuario.

Cuarto Hito: liberación del producto software para el cliente.

E. Observaciones sobre ICONIX

Rosenberg y Scott (1999), Rosenberg y Scott (2001) y Rosenberg y Stephens (2007), advierten sobre problemas y dudas comunes a los equipos que desarrollan software. Están relacionadas con cada técnica del modelado, como indicamos:

a. Modelo de dominio.- (1) No perder mucho tiempo con la verificación gramatical, (2) No adicionar multiplicidad muy pronto al proyecto, (3) Adicionar agregación y composición sólo en el diseño detallado, (4) No diseñar un modelo de dominio inicial con más de 7-9 clases;

b. Modelo de caso de uso.- (1) No escribir casos de uso sin saber que hacen realmente los usuarios, (2) No perder tiempo haciendo modelos elegantes de casos de uso que no sirven para construir el diseño, (3) No perder tiempo discutiendo donde usar include o extend, (4) No escribir plantillas de casos de uso largos o complejos;

c. Análisis de robustez.- (1) No hacer diseño detallado en el diagrama de robustez, (2) No perder tiempo perfeccionando el diagrama de robustez

a medida que el diseño evoluciona;

d. Diagrama de secuencia.- (1) No asignar comportamiento a los objetos sin saber realmente que hacen los objetos, (2) No comenzar a diseñar el diagrama de secuencia antes de completar el diagrama de robustez relacionado, (3) No enfocar su atención en definir los métodos get y set, en perjuicio de los métodos reales.

G. Técnicas de la Metodología ICONIX

G.1 Modelado de Dominio

Rosenberg y Scott (1999), Rosenberg y Stephens (2007) definen el modelado de dominio como, la tarea de descubrir objetos (clases), que representan cosas y conceptos del "mundo real". El modelo de dominio para un proyecto define el alcance y la base para construir los casos de uso. El modelo de dominio sirve como un glosario de términos, que es utilizado al inicio para escribir los casos de uso. El objetivo del modelado de dominio es realizar un primer levantamiento de las clases (objetos) que forman parte del problema.

De acuerdo a Rosenberg y Stephens (2007), el enfoque de ICONIX asume que el modelo de dominio inicial es incorrecto y proporciona un mejoramiento incremental a medida que se analizan los casos de uso.

Rosenberg y Scott (1999) definen una clase como "una descripción de un conjunto de objetos con propiedades similares, comportamiento común, relaciones comunes y semántica común".

Rosenberg y Stephens (2007), alcanzan diez consejos para el modelado de dominio, listado en orden a su importancia, siendo:

1. Enfocarse en los objetos del mundo real (dominio del problema);
2. Usar las relaciones de generalización (es-un) y agregación (tiene-un) para mostrar las relaciones de los objetos entre sí;
3. Limitar sus esfuerzos para el modelado de dominio inicial a dos horas;
4. Organizar sus clases con abstracciones clave en el dominio del problema;
5. No confundir el modelo de dominio con el modelo de datos;
6. No confundir un objeto con una tabla de la base de datos;

7. Usar el modelo de dominio como un glosario del proyecto;
8. Hacer su modelo de dominio inicial antes de escribir sus casos de uso, para evitar ambigüedad de nombres;
9. No espere que su diagrama de clases final coincida con su modelo de dominio, pero recuerde que debe existir cierto parecido entre ellos;
10. No ponga clases interface y control en su modelo de dominio.

Los errores más frecuentes del modelado de dominio, destacados por Rosenberg y Scott (1999), son:

1. Asignar multiplicidad a las asociaciones de clases al inicio del modelado;
2. Analizar los verbos y sustantivos exhaustivamente;
3. Asignar operaciones a las clases sin explorar los casos de uso y diagramas de secuencia;
4. Debatir sobre usar agregación o composición para cada asociación;
5. Presumir una estrategia de implementación específica en esta actividad;
6. Usar nombres difíciles de entender para las clases;
7. Hacer directamente la implementación;
8. Crear un esquema de "uno-a-uno" entre las clases del modelo de dominio y las tablas de la base de datos;
9. Utilizar patrones de diseño de forma prematura.

G.2 Modelado de Casos de Uso

El modelo de casos de uso es el centro conceptual del desarrollo, porque guía todo el proceso ICONIX. Así, Rosenberg y Scott (1999) muestran elementos clave, donde; los casos de uso son desarrollados a partir del modelo de dominio, el análisis de robustez identifica un conjunto de objetos que satisfacen cada caso de uso, el diagrama de secuencia traza el flujo de los mensajes entre los objetos conforme se especifico en los casos de uso, los requisitos de usuario son asignados a los casos de uso y clases, los casos de uso se usan para las pruebas durante la implementación.

Un caso de uso describe la interacción entre el usuario y el sistema para alcanzar un objetivo (Rosenberg y Stephens, 2007). Un caso de uso describe y valida lo que el sistema debe hacer, sirve para el control entre el usuario,

cliente y desarrolladores.

"El caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso (Larman, 2000; ápud Jacobson, 1992). Los casos de uso son historias o casos de utilización de un sistema, no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen tácitamente los requerimientos en las historias que narran". (Larman, 2000)

Es importante indicar que UML (OMG®, 2001) no define un estándar para describir casos de uso. Esta actividad debe ser definida por el flujo de trabajo de la metodología seleccionada.

Los actores representan el rol de una entidad externa al sistema, como: un usuario, el hardware, otro sistema que interactúa con el sistema modelado. Los actores no forman parte del sistema, pero identifican sus límites. Fowler y Scott (2000) afirman que, cuando hablamos de actores, es importante pensar en los roles y no en las personas o en cargos. Un único actor puede desempeñar varios casos de uso, y para un caso de uso puede haber muchos actores.

Los actores son clases, un actor está relacionado a uno o más casos de uso mediante asociaciones y, puede poseer relaciones de generalización.

Un paquete en UML (OMG®, 2001), es un elemento organizacional. Permite agregar diferentes elementos de un sistema en grupos, de forma que semánticamente o estructuralmente tenga sentido. Entonces, un grupo de casos de uso relacionados será definido como paquete de casos de uso.

Rosenberg y Stephens (2007), presentan diez consejos para el modelado de casos de uso, los ítems del 1 al 6 describen el uso del sistema y del 7 al 10 para describir los casos de uso en el contexto del modelo de dominio, como sigue:

1. Siga la regla de dos párrafos (curso básico y curso alterno);
2. Organizar sus casos de uso con actores y diagramas de caso de uso;
3. Escriba sus casos de uso en voz activa (las oraciones activas dejan en

claro quién hace que y, escriba desde la perspectiva del usuario);

4. Escriba sus casos de uso utilizando un flujo de evento/respuesta (cómo el usuario utiliza el sistema y que responde el sistema), describiendo ambos lados del dialogo usuario/sistema;
5. Use prototipos GUI e historia de eventos del usuario;
6. Recordar que sus casos de uso son especificaciones del comportamiento del sistema en tiempo de ejecución;
7. Escriba sus casos de uso en el contexto del modelo de dominio (objetos);
8. Escriba sus casos de uso utilizando una estructura tipo "sustantivo-verbo-sustantivo";
9. Haga referencia a los objetos de dominio por su nombre;
10. Haga referencia a las clases interfaz por su nombre.

Describir los casos de uso en el formato "sustantivo-verbo-sustantivo", los sustantivos son los objetos, éstos vienen del modelo de dominio (objetos entidad) o son objetos interfaz. Los verbos son los mensajes entre objetos y, representan las funciones del software (controladores) que deben ser construidos.

Diez errores que deben evitarse cuando escribe los casos de uso, destacados por Rosenberg y Scott (1999), son:

1. Escribir requisitos funcionales en lugar del escenario de uso;
2. Describir atributos o métodos en lugar de uso;
3. Escribir los casos de uso de forma muy simplificada;
4. Desvincular la descripción de la interface del usuario;
5. Evitar nombres explícitos para los objetos interfaz;
6. Escribir en voz pasiva (ejemplo, el verbo "ser/estar" seguido de un verbo en tiempo pasado, es señal clara de una oración pasiva);
7. Describir solo interacciones de usuario, ignorando respuestas del sistema;
8. Omitir la descripción del curso alterno;
9. Describir algo que no es parte del caso de uso, como; precondiciones o post-condiciones;
10. Perder tiempo decidiendo si usar includes o extends.

No generalizar casos de uso, porque es algo como "generalizar el manual del usuario", seguro dirá ¿Qué cosa?. (Rosenberg y Stephens, 2007)

Relación	Descripción	Solución
A <include> B	A medio camino del caso de uso A, se llama al caso de uso B. Cuando B termina, A se lleva a cabo desde que B se detiene. Es similar a la llamada de una función o GOSUB en BASIC. ➤Es como decir "A tiene un B".	Bala de plata
A <extend> B	Todos los pasos del caso de uso A se realizan durante la ejecución del caso de uso B, en el punto de extensión que se especifica en B. ➤ En la mayoría de los casos <extends> es <includes> con una flecha hacia atrás. (Ambos son subtipos de <invokes>).	Estaca en el corazón
A <precedes> B	El caso de uso A debe llevarse a cabo en su totalidad antes de comenzar el caso de uso B	Agua bendita
A <invokes> B	El caso de uso B sucede durante la vida útil del caso de uso A.	Una buena taza de té y un huevo de pascua.

Tabla N° 2.11: Relaciones comunes de casos de uso. (Rosenberg y Stephens, 2007)

Casos de Uso	Algoritmo
Diálogo entre usuario y sistema	Computación "atómica"
Secuencia de evento/respuesta	Serie de pasos
Cursos básico/alterno	Un paso de un caso de uso
Múltiples objetos participantes	Operación en una clase
El usuario y sistema	Todo el sistema

Tabla N° 2.12: Casos de uso versus algoritmos. (Rosenberg y Stephens, 2007)

G.3 Revisión de Requisitos (Primer Hito)

Para garantizar que el sistema coincide con los requisitos. La revisión de requisitos debe incluir al cliente, el analista de negocios, un desarrollador senior, un usuario final y, un administrador del proyecto.

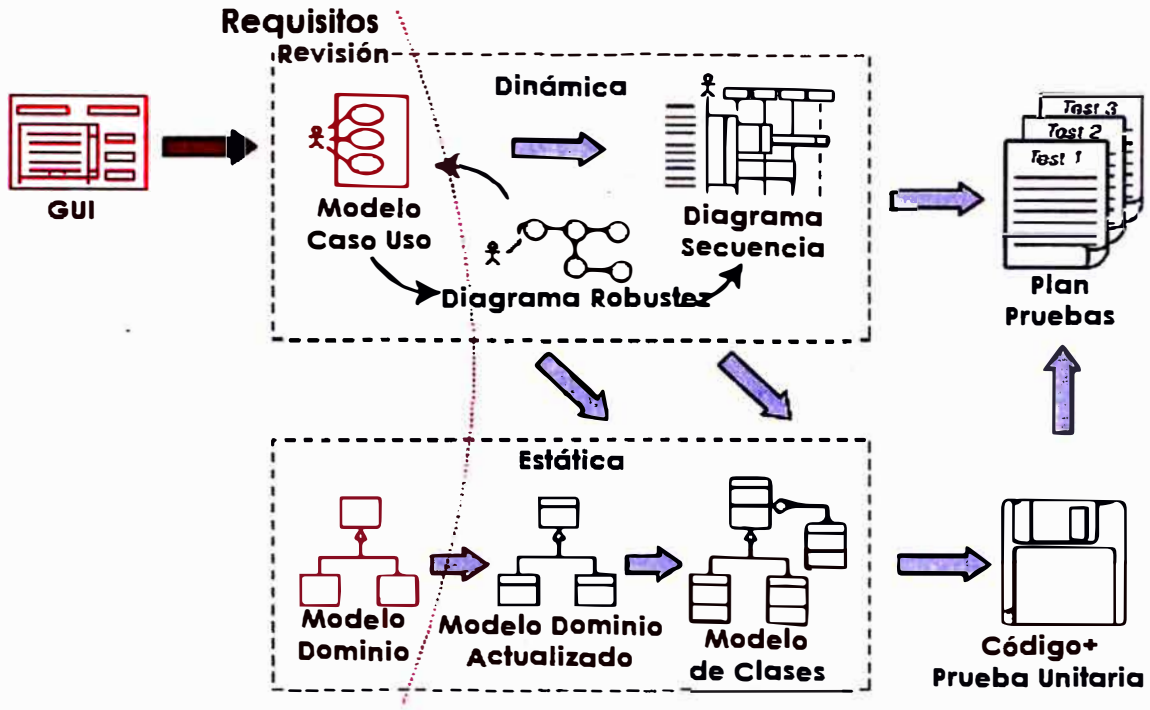

Figura N° 2.9: ICONIX - Revisión de requisitos. (Rosenberg y Stephens, 2007)

Diez consejos para la revisión de requisitos de acuerdo a Rosenberg y Stephens (2007), que mostramos a continuación:

1. Asegurar que el modelo de dominio contiene al menos 80% de objetos más importantes del dominio de problema (objetos del mundo real);
2. Asegurar que el modelo de dominio muestra relaciones de generalización y agregación entre los objetos de dominio;
3. Asegurar que los casos de uso describen el curso básico y curso alterno, en voz activa;
4. Asegúrese que la voz pasiva y los requisitos funcionales, no están en la descripción de los caso de uso;
5. Asegúrese que ha organizado los casos de uso en paquetes y cada paquete tiene un diagrama de caso de uso;
6. Asegúrese que sus casos de uso están escritos en el contexto del modelo de dominio;
7. Los casos de uso deben estar escritos en contexto de interfaz de usuario;
8. Asegúrese que ha completado la descripción del caso de uso con alguna historia o prototipo GUI;
9. Revisar los casos de uso, modelo de dominio y prototipos GUI con el personal indicado para esta actividad;

10. Revisar los casos de uso mediante los "ocho pasos sencillos para un mejor caso de uso" como se indica:
    I. Eliminar todo lo que este fuera del ámbito de aplicación;
    II. Cambiar la descripción de voz pasiva a voz activa;
    III. Compruebe que la descripción no es demasiado abstracta;
    IV. Presentar con precisión la GUI relacionada;
    V. Nombre los objetos de dominio que participan;
    VI. Asegúrese que tiene todos los cursos alternos;
    VII. Asocie cada requisito a los casos de uso;
    VIII. Describir lo que el usuario intenta hacer para cada caso de uso.

G.4 Análisis de Robustez

El concepto de análisis de robustez fue introducido por Ivar Jacobson para el mundo de la orientación a objetos en 1991. El análisis de robustez es, analizar los casos de uso e identificar un primer conjunto de objetos para cada caso de uso (Rosenberg y Scott, 1999), observamos el esquema de la actividad en la figura 2.10, los objetos son clasificados en tres estereótipos:

a. Objeto interfaz.- capa de presentación que los actores usan para interactuar con el sistema (pantallas o páginas Web), objeto en el caso de uso que debe tener un nombre (sustantivo);
b. Objeto entidad.- son objetos del modelo de dominio, objeto en el caso de uso que debe tener un nombre (sustantivo);
c. Objeto control (controlador).- funcionan como "pegamento" entre los objetos interfaz y los objetos entidad, éste objeto en el caso de uso debe ser un verbo. Un controlador en un diagrama de robustez no siempre es una clase control real, puede ser contenedor de una función de software y, son convertidos en métodos de los objetos entidad o interfaz.

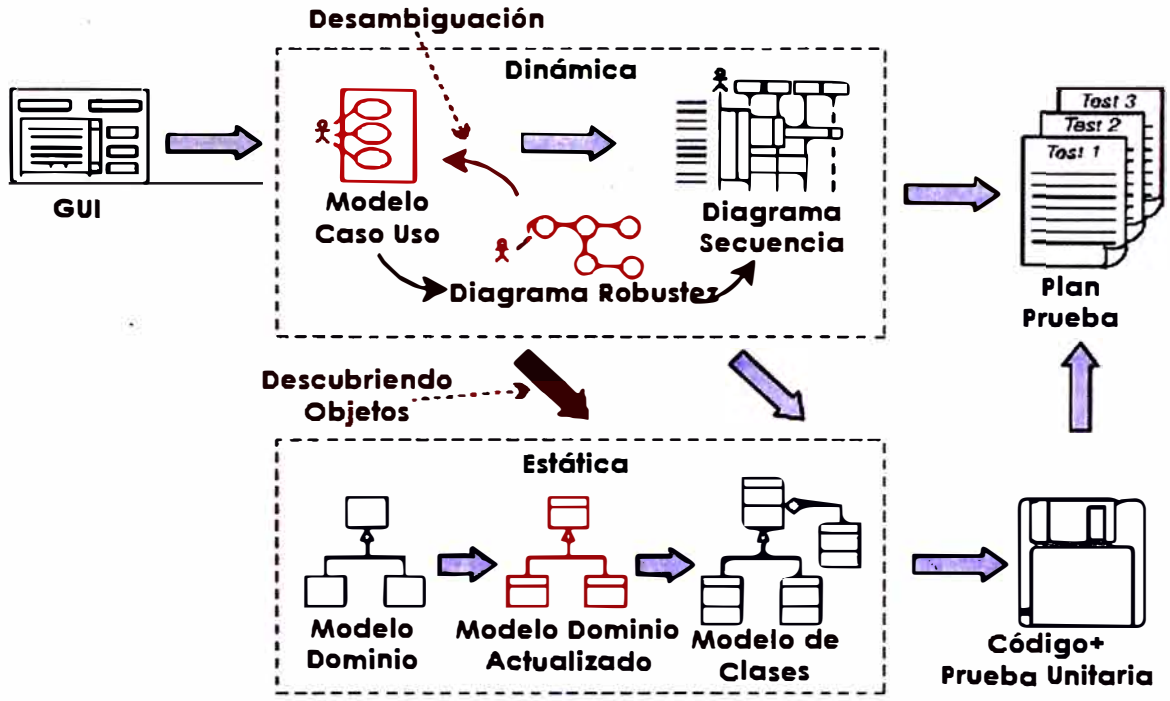

Figura N° 2.10: ICONIX – Análisis de robustez. (Rosenberg y Stephens, 2007)

El diagrama de robustez es una "imagen" de un caso de uso, el diagrama de robustez y la descripción del caso de uso deben coincidir con precisión. Si el análisis (casos de uso) es el "Qué" y diseño el "Cómo", entonces el análisis de robustez es un diseño preliminar. Por lo tanto, es parte del análisis y del diseño.

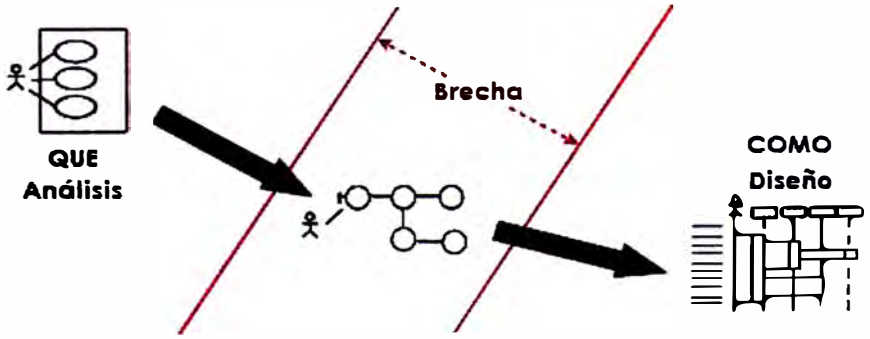

Figura N° 2.11: ICONIX – Análisis de robustez entre análisis y diseño. (Rosenberg y Stephens, 2007)

La descripción del caso de uso en el formato nombre-verbo-nombre, permite construir el diagrama de robustez y el diseño detallado fácilmente, el análisis de robustez ayuda a desambiguar el caso de uso, ver la figura 2.10.

Reglas que ayudan a hacer cumplir el formato nombre-verbo-nombre en la

descripción del caso de uso, para construir los diagramas de robustez, son:

a. Los nombres (objetos entidad e interfaz) pueden comunicarse con los verbos (y viceversa).
b. Los nombres no pueden comunicarse con otros nombres.
c. Los verbos (controladores) pueden comunicarse con otros verbos.

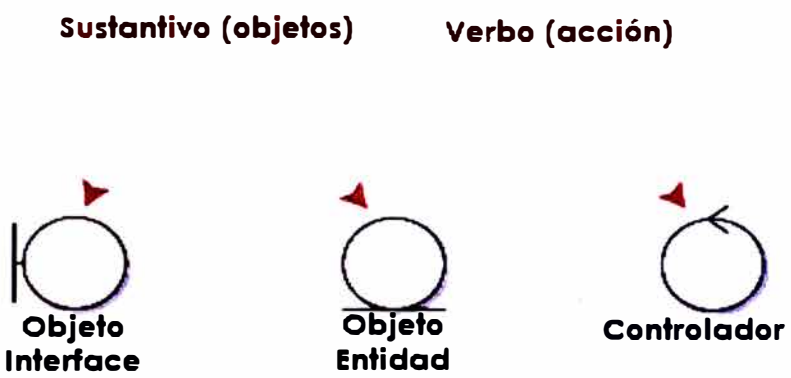

Figura N° 2.12: ICONIX – Símbolos del diagrama de robustez. (Rosenberg y Stephens, 2007)

En el diagrama de robustez, se describe los elementos del GUI usando los objetos interfaz, las funciones de software usando los controladores y, los objetos de dominio usando las entidades.

Diez normas para construir el diagrama de robustez según Rosenberg y Stephens (2007), indicamos a continuación:

1. Pegar la descripción (usara del inicio a fin, línea por línea) del caso de uso en la interface para construir el diagrama de robustez;
2. Usar las clases entidad del modelo de dominio, para actualizar el modelo de dominio al descubrir clases durante el análisis de robustez y construir el diagrama de robustez;
3. Rescribir los casos de uso mientras hace el análisis de robustez (técnica de desambiguación);
4. Crear un objeto interfaz para cada pantalla y nombrar esas pantallas sin ambigüedad;
5. Recordar que los controladores a veces no son objetos control real, normalmente son funciones de software lógicas.
6. No preocuparse sobre la dirección de las flechas en un diagrama de

robustez.

7. Relacionar un caso de uso a un diagrama de robustez, solo si este es invocado de otro caso de uso;
8. El diagrama de robustez representa el diseño conceptual (diseño preliminar), el diseño real es mostrado en el diagrama de secuencia;
9. Las clases interfaz y entidad de un diagrama de robustez, se convertirán en objetos en un diagrama de secuencia, mientras que los controladores se convertirán en mensajes;
10. Un diagrama de robustez es una "imagen" de un caso de uso, cuyo fin es refinar la descripción del caso de uso y modelo de dominio (actualizado).

Los diez beneficios del análisis de robustez, presentados por Rosenberg y Scott (1999), son:

1. Obliga la escritura de los casos de uso de forma correcta y concreta;
2. Fuerza la escritura de los casos de caso en voz activa;
3. Provee mecanismos de verificación de los casos de uso;
4. Ayuda a definir reglas de sintaxis del tipo "el actor interactúa solo con objetos interfaz", para los casos de uso;
5. Construir un diagrama de robustez es más fácil y más rápido, que un diagrama de secuencia;
6. Permite esbozar un framework para GUI-lógica-datos para sistemas cliente/servidor;
7. Permite relacionar lo que hará el sistema (casos de uso) y como funcionará el sistema (diagrama de secuencia);
8. Llena el vacío entre el análisis (caso de uso) y diseño (diagrama de secuencia);
9. Permite identificar el reuso de estructuras definidas en la realización de los casos de uso;
10. Facilita el uso del paradigma, modelo - vista - control.

Según Rosenberg y Stephens (2007), antes de pasar a la revisión del diseño preliminar, hay que actualizar el diagrama de clases continuamente, mientras se trabaja con los casos de uso y el análisis de robustez.

G.5 Revisión de Diseño Preliminar (Segundo Hito)

Para asegurarse que los diagramas de robustez, el modelo de dominio y, la descripción de casos de uso coincidan entre sí. Esta revisión es el "puente" entre el diseño preliminar y el diseño detallado, se hace para cada paquete de casos de uso. Las personas involucradas en esta actividad son, el mismo grupo de la revisión de requisitos.

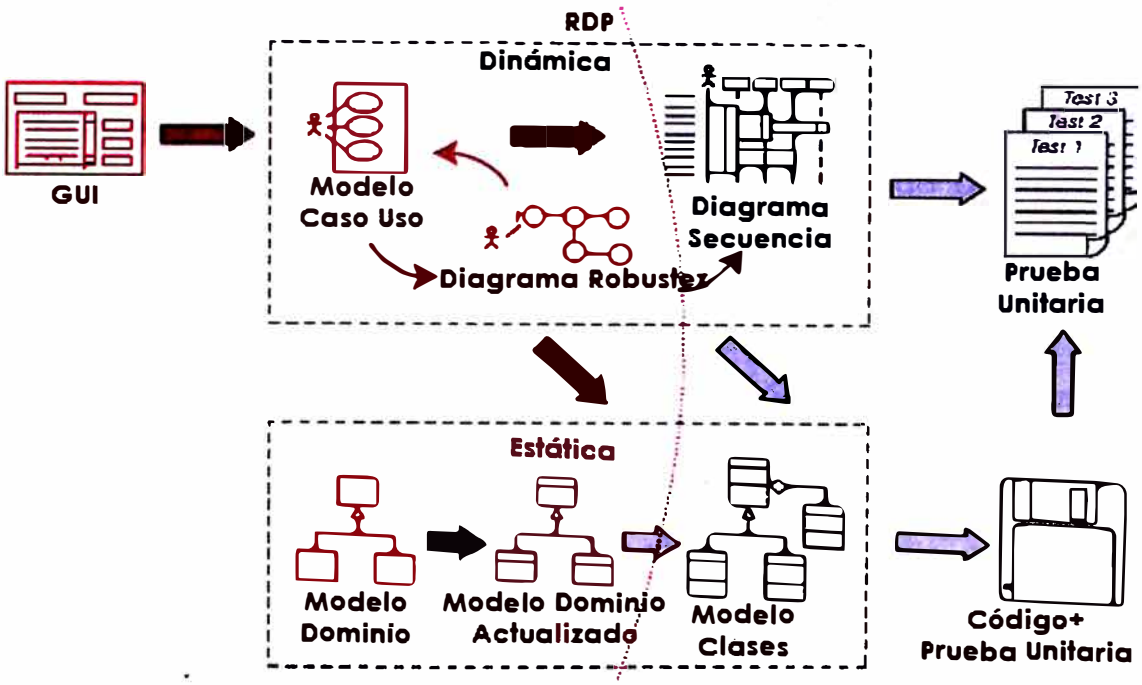

Figura N° 2.13: ICONIX – Revisión de diseño preliminar. (Rosenberg y Stephens, 2007)

Las reglas que permiten hacer una buena revisión del diseño preliminar, son principios en resumen, como una lista de consejos. (Rosenberg y Scott, 2001)

1. Para cada caso de uso asegúrese que su descripción coincide con el diagrama de robustez, utilizar la prueba del resaltador;
2. Asegúrese que todas las clases entidad de todos los diagramas de robustez aparecen en el modelo de dominio actualizado;
3. Asegúrese que puede trazar flujos de datos entre clases entidad y clases interfase;
4. No olvide escribir los cursos alternos y su comportamiento, para cada caso de uso;
5. Asegúrese que la descripción de cada caso de uso, cubre ambos lados del diálogo entre usuario y sistema;
6. Use correctamente las reglas de sintaxis de comunicación entre objetos

para construir el diagrama de robustez;

7. Asegúrese que esta revisión incluye a personal no-técnico (clientes, equipo de marketing, etc.) y técnico (arquitectos de sistema, programadores, etc.);
8. Asegúrese que los casos de uso están en el contexto del modelo de objetos y en el contexto de la interfaz gráfica;
9. Asegúrese que el diagrama de robustez y su descripción del caso de uso, no están mostrando el nivel de detalle del diagrama de secuencia;
10. Siga los "seis pasos sencillos" para el mejor diseño preliminar, pasos ampliados en Rosenberg y Scott (2001), aplicar a los diagramas de robustez y descripción de los casos de uso, para cada diagrama de robustez, hacer lo siguiente:
    - El diagrama debe coincidir con la descripción del caso de uso;
    - El diagrama debe cumplir las reglas del análisis de robustez;
    - Comprobar que el diagrama sigue el flujo lógico del caso de uso;
    - El diagrama debe tener todos los cursos alternos para el caso de uso;
    - Tener cuidado con los "diseños de patrones " en el diagrama;
    - Verificar que el diagrama no es un diseño detallado.

G.6 Diagrama de Secuencia

Finalizado el análisis de robustez y la revisión de diseño preliminar, podemos comenzar el diseño detallado (asignación de comportamiento). La descripción de los casos de uso debe ser completa, correcta, detallada y explícita (casos de uso para crear un diseño detallado), tener descubiertas casi todas las clases de dominio y, contar con la arquitectura técnica.

El comportamiento de un caso de uso se detalla mediante el diagrama de secuencia, éste muestra la colaboración dinámica entre objetos del sistema, también observamos la secuencia de mensajes enviados entre los objetos (interacción entre los objetos). El diagrama de secuencia, ayuda ha asignar operaciones a los objetos interfaz y entidad, porque los controladores (verbos) del diagrama de robustez, se convierten en mensajes (operaciones) entre los objetos interfaz y entidad (nombres). En conclusión, actualizamos el modelo estático con las operaciones y los atributos identificados en las fases anteriores.

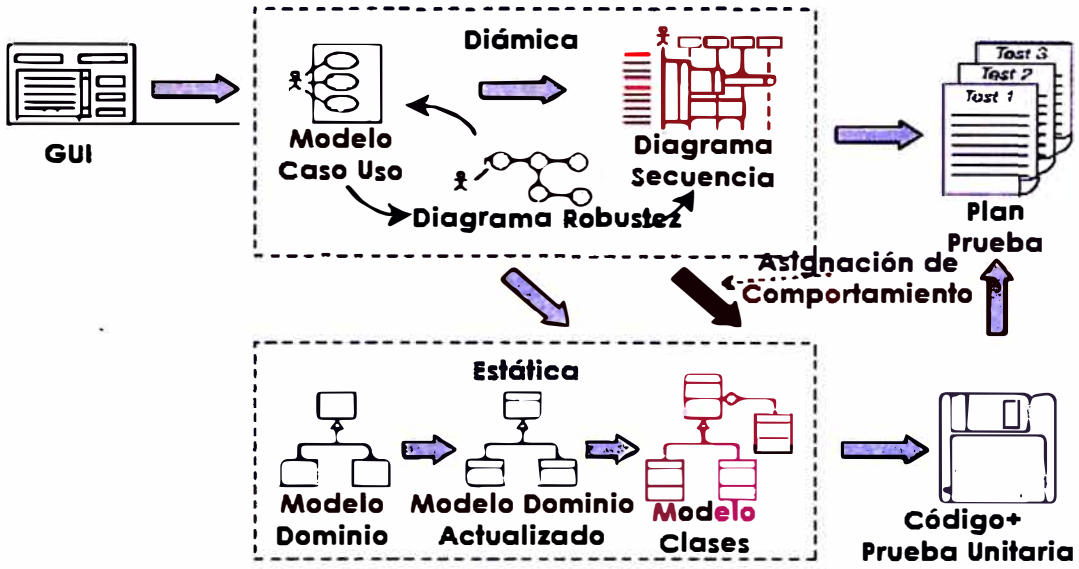

Figura N° 2.14: ICONIX - Diagrama de secuencia. (Rosenberg y Stephens, 2007)

Según Rosenberg y Scott (1999) pasos construir el diagrama de secuencia, son:

1. Copiar texto del caso de uso al margen izquierdo;
2. Objetos del diagrama de robustez, presentar el nombre del objeto, opcionalmente el nombre de la clase (objeto::clase);
3. Mensajes, representados como flechas entre los objetos;
4. Métodos, son la implementación de las operaciones.

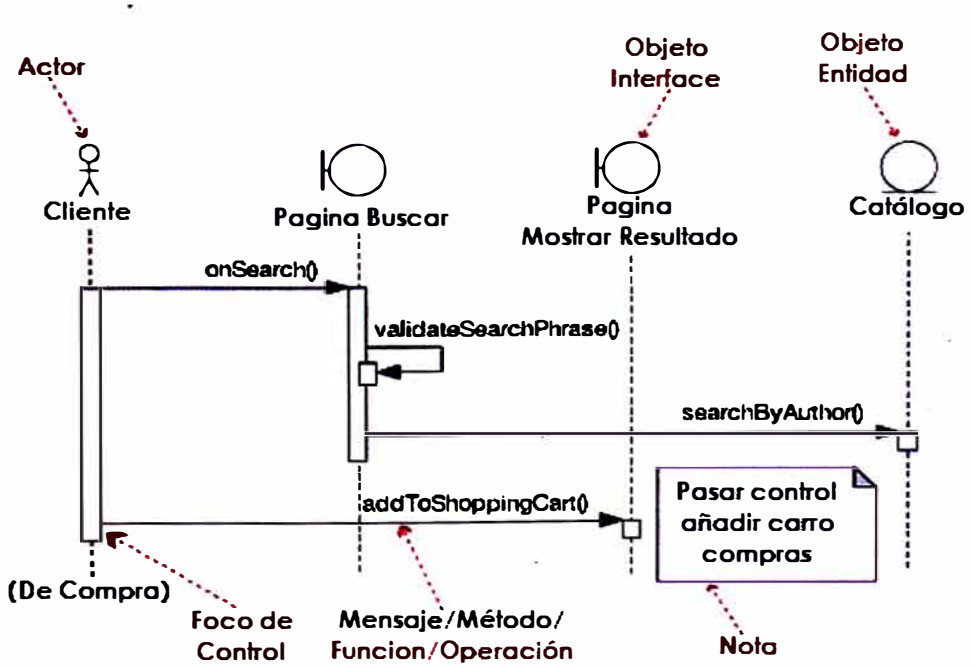

Figura N° 2.15: ICONIX - Elementos del diagrama de secuencia. (Rosenberg y Stephens, 2007)

Decidir que mensajes y clases quedan, es la esencia del modelado de secuencia, no es tarea fácil y exige mucho esfuerzo y experiencia (Rosenberg y Scott, 1999). ICONIX presenta algunas sugerencias para esta tarea:

a. Comenzar convirtiendo los controladores del diagrama de robustez, como mensajes que representan el comportamiento deseado;
b. Usar el diagrama de robustez como una lista de chequeo, para estar seguros que todo el comportamiento necesario del sistema está en el diagrama de secuencia;
c. Responder a la pregunta "qué objetos son responsables de que funciones?";
d. Diseñar mensajes entre objetos es equivalente a atribuir operaciones para las clases.

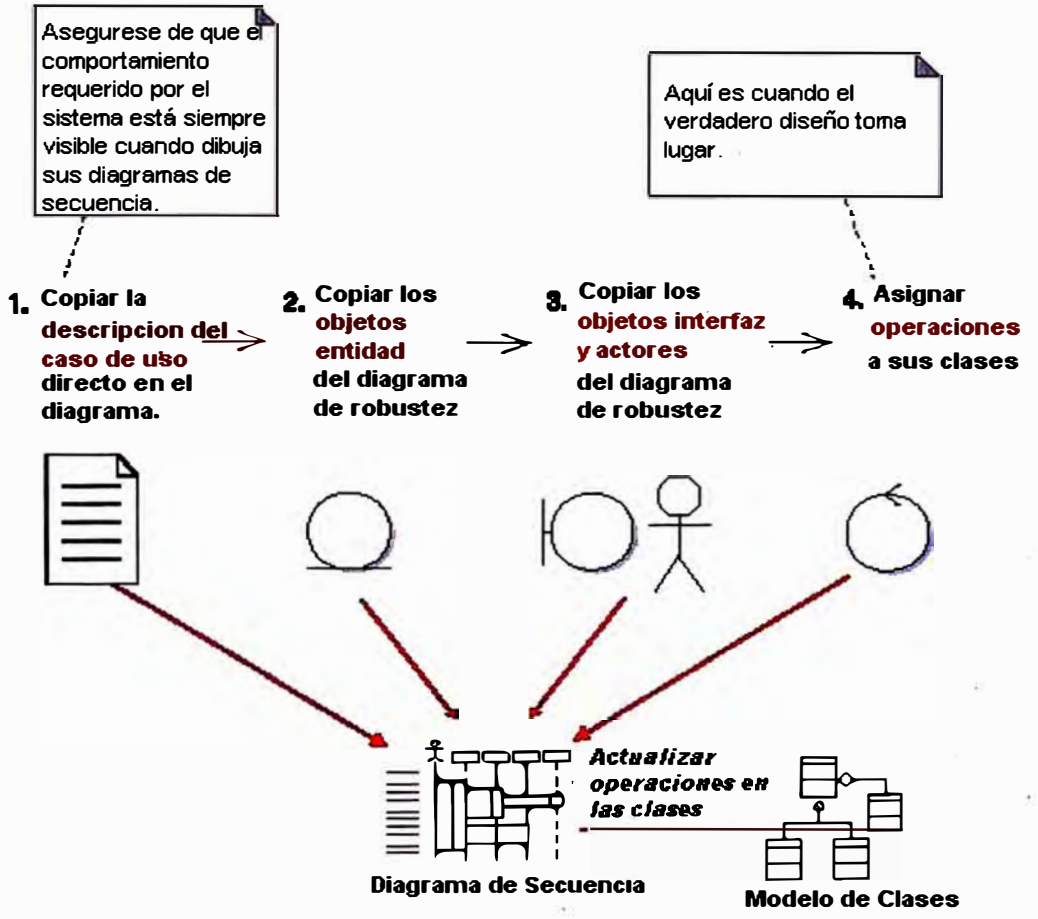

Figura N° 2.16: ICONIX – Cuatro pasos para construir el diagrama de secuencia.

Las diez reglas más importantes para construir el diagrama de secuencia,

sugeridos por Rosenberg y Stephens (2007), son:

1. Comprender porque se esta dibujando un diagrama de secuencia;
2. Dibujar un diagrama de secuencia para cada caso de uso, incorpore todo el curso básico y los cursos alternos;
3. Comenzar el diagrama de secuencia con las clases interfaz, entidad, actores y, la descripción de los casos de uso del análisis de robustez;
4. Usar èl diagrama de secuencia para mostrar cómo el comportamiento del caso de uso (controladores del diagrama de robustez) es realizado por los objetos;
5. Asegúrese que la descripción de los casos de uso tienen relación con los mensajes del diagrama de secuencia;
6. No perder mucho tiempo preocupándose por la línea de vida del objeto;
7. Asignar operaciones a las clases mientras dibuja los mensajes;
8. Revisar el diagrama de clases mientras asigna sus operaciones, para estar seguro que las operaciones están en las clases que le corresponden;
9. Hacer refactoring al diseño en el diagrama de secuencia antes de codificar;
10. Limpiar el modelo estático antes de la revisión critica de diseño, para; solucionar los problemas del diseño del mundo real, identificar los patrones útiles que puedan mejorar el diseño.

De acuerdo con Rosenberg et al. (2005), para completar el diseño detallado, actualizar el diagrama de clases usando el diagrama de secuencia, debe adicionar operaciones, todos los atributos, visibilidad y navegabilidad a las clases.

G.7 Revisión Crítica de Diseño (Tercer Hito)

Hacer esta actividad, cuando tiene todos los diagramas de secuencia y, antes de codificar, objetivos; los requisitos (qué) deben coincidir con el diseño detallado (cómo) (cada caso de uso debe coincidir con su diagrama de secuencia), revisar la calidad del diseño, la dirección de las flechas de mensajes entre objetos del diagrama de secuencia debe ser correcta.

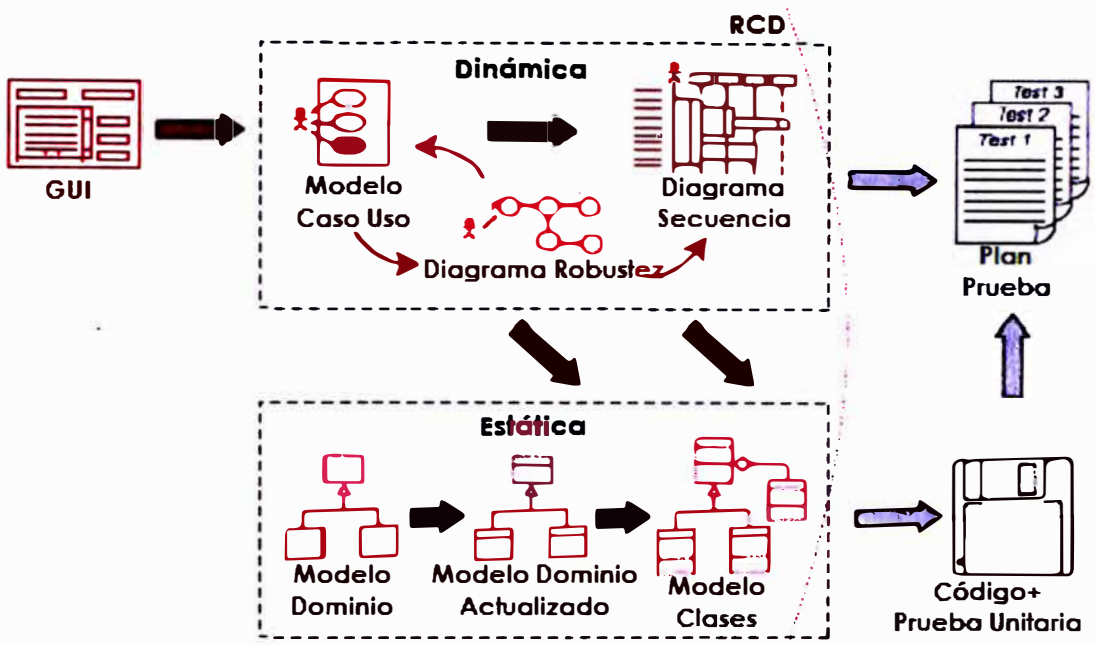

Figura N° 2.17: ICONIX – Revisión critica de diseño. (Rosenberg y Stephens, 2007)

Para Rosenberg y Stephens (2007), existen diez elementos clave para la revisión crítica de diseño, que son:

1. Verificar que los diagramas de secuencia coincidan con la descripción de los casos de uso;
2. Verificar (sí, otra vez) que cada diagrama de secuencia representa los cursos básicos y alternos de acción;
3. Verificar que las operaciones se asignaron correctamente a las clases;
4. Revisar las clases del diagrama de clase y ver que todas tienen atributos y operaciones apropiadas.
5. Si el diseño usa patrones u otras construcciones detalladas de aplicación, verificar que estos detalles se reflejan en el diagrama de secuencia;
6. Relacione los requisitos funcionales (no funcionales) a los casos de uso y a las clases, para asegurarse que fueron considerados todos;
7. Verificar que los programadores "analizaron" el diseño y que pueden construirlo para que funcione como se ha previsto;
8. Verificar que todos los atributos son correctos, sus valores de retorno y la lista de parámetros en sus operaciones son completas y correctas;
9. Generar la cabecera de código de las clases e inspecciónalo;
10. Revisa el plan de prueba para esta versión.

G.8 Implementación

El proceso seguido hasta ahora se aplica para codificar en cualquier lenguaje de programación orientado a objetos. Según ICONIX, el diseño no se entrega a los programadores, éstos deben estar involucrados en el proceso de diseño, para que estén informados sobre los detalles de la implementación.

El objetivo de la implementación según ICONIX, es cómo hacer el último salto del diseño detallado al código fuente y, mostrar cómo encaja todo para comenzar a codificar. Antes de comenzar la codificación del diseño, definir la arquitectura técnica, contar con los diagramas de secuencia para cada caso de uso y, haber completado la revisión crítica de diseño.

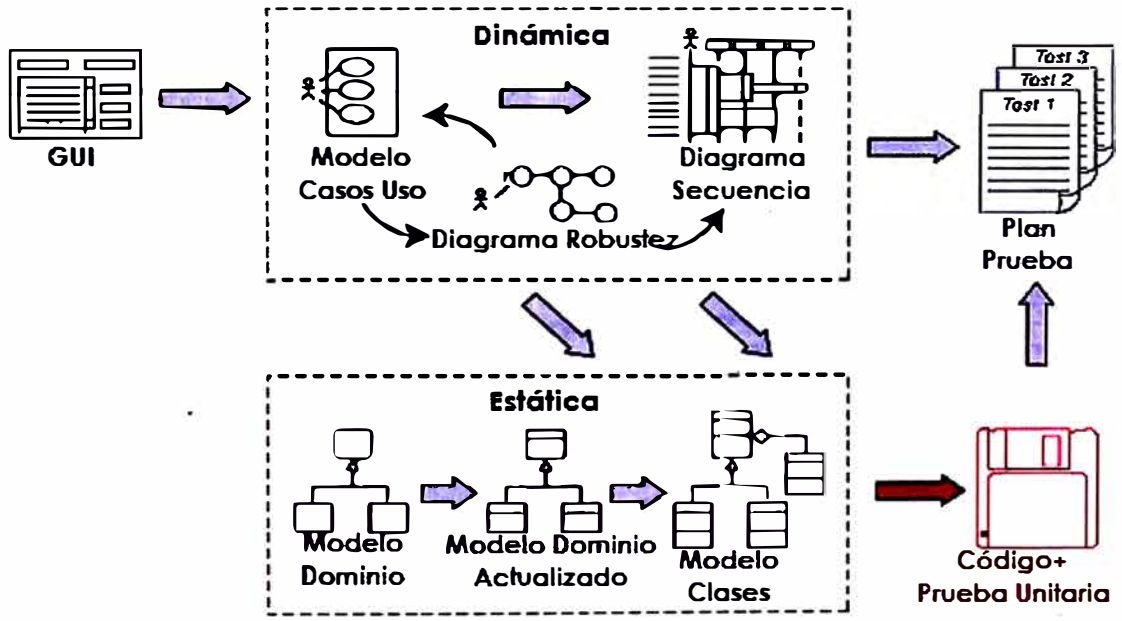

Figura N° 2.18: ICONIX – Implementación. (Rosenberg y Stephens, 2007)

Presentamos los principios más importantes para la implementación, recomendados por Rosenberg y Stephens (2007), son:

1. Conducir la codificación directamente desde el diseño;
2. Si la codificación revela que el diseño esta mal, cámbialo y revisa el proceso;
3. Revisar periódicamente el código;
4. Cuestionar siempre las opciones de diseño del framework;
5. No deje que el framework se haga cargo de la capa del negocio;
6. Si el código comienza a salirse de control, para y revisa el diseño;

7. Mantener sincronizados el diseño y el código;
8. Centrarse en las pruebas unitarias, mientras implementa el código;
9. No comentar excesivamente el código, hace que el código sea menos mantenible y difícil de leer;
10. Recordar implementar los cursos alternos y los cursos básicos.

G.9 Pruebas

Las prueba basadas en diseño proveen un método para producir casos de prueba desde los casos de uso y verificar que los escenarios se implementaron correctamente. Las pruebas deben estar estrechamente ligadas a los requisitos, para verificar que cada requisito ha sido implementado correctamente. Comenzar las pruebas identificando los casos de prueba desde los diagramas de robustez, luego escribir el código para pruebas unitarias durante la implementación.

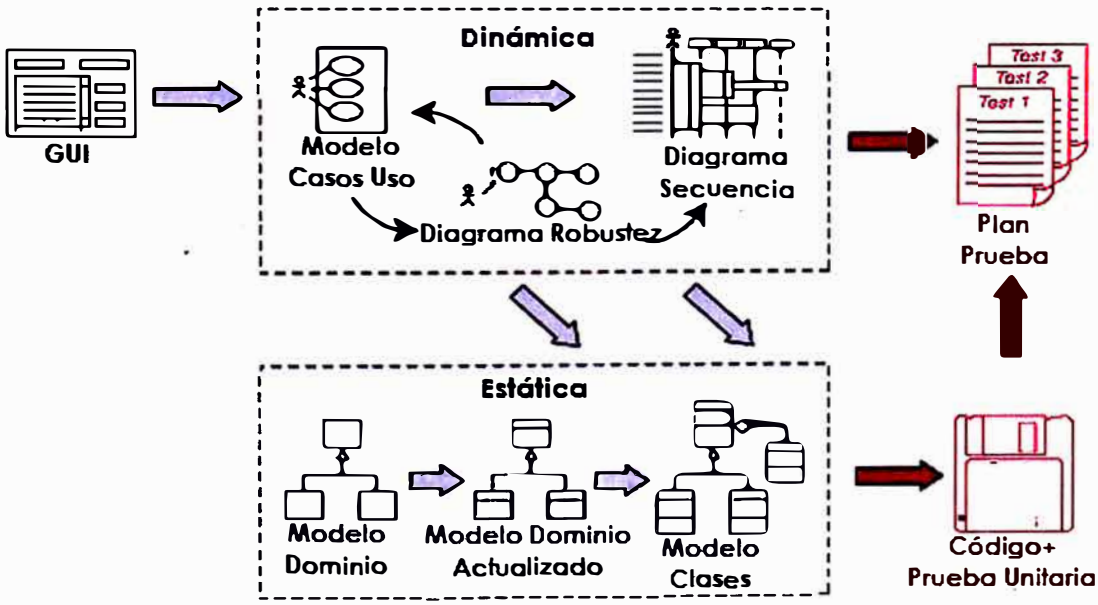

Figura N° 2.19: ICONIX – Pruebas basadas en diseño. (Rosenberg y Stephens, 2007)

Prueba	¿Cuándo Aplicar?
Prueba unitaria.- Prueba para componentes individuales de software (controladores, operación). El resultado puede usarse para simular entradas y salidas de un componente de forma que opere en modo autónomo	La prueba unitaria se ejecuta en cada integración del software durante el desarrollo (incluso en corrección de errores cuando el software es derivado a pruebas de sistema)
Prueba de aceptación.- Prueba conducida por el cliente para	La prueba de aceptación opera en línea con la prueba del sistema (para probar

determinar si el sistema reúne los requisitos especificados en el contrato	que se entrega el sistema que fue especificado)

Tabla N° 2.13: Las pruebas y su aplicación. (Sociedad Computacional Británica, 2007)

Las recomendaciones más importantes para las pruebas, según Rosenberg y Stephens (2007), son:

1. "Pensar en las pruebas", si encuentras y corriges el error durante la prueba, los usuarios no lo encontrarán en el producto final.
2. Prepara cada prueba por adelantado y, aplica cada prueba en el momento correcto;
3. Al realizar pruebas unitarias, crea una prueba para cada controlador de los diagramas de robustez. Además, crea una prueba unitaria para cada operación de las clases de diseño. A veces, son lo mismo (un controlador es implementado como una operación simple en una clase), y otras veces un controlador se realiza mediante varias operaciones;
4. Para sistemas de tiempo real, usa los diagramas de estado como base para los casos de prueba;
5. Verifica a nivel de requisitos, cada requisito debe ser explicado;
6. Construye una matriz para asistir la verificación de requisitos;
7. Realiza pruebas de aceptación para cada escenario de los casos de uso;
8. Diseñar pruebas para cubrir una ruta completa de escenarios desde un curso básico;
9. Usa una herramienta, como JUnit, para almacenar y organizar las pruebas unitarias;
10. Realiza mantenimiento a tus pruebas unitarias de manera regular.

2.2.3 EL PROCESO PROGRAMACION EXTREMA (XP)

Extreme Programming (XP) es ligera, eficiente, flexible y de bajo riesgo para equipos pequeños y medianos, que desarrollan software con requisitos dinámicos o en constante cambio. (Beck, 1999)

"Extreme Programming es una forma de desarrollar software basado en los valores de simplicidad, comunicación, retroalimentación y coraje. Funciona haciendo trabajar a todo el equipo sobre un conjunto de prácticas sencillas, con suficiente retroalimentación para ver dónde se encuentran y cómo

aplicar dichas prácticas a cada situación particular". (Jeffries, 2001)

De acuerdo a Beck (1999), extreme programming se diferencia de otros procesos por:

a. Presentar realimentación continua y concreta en ciclos cortos;
b. Aplicar planificación incremental, al inicio un plan global, que evoluciona durante el ciclo de vida del proyecto;
c. Tener flexibilidad para implementar la funcionalidad, respondiendo a los cambios de las reglas de negocio;
d. Confiar en pruebas automatizados escritos por programadores y clientes, para monitorear el desarrollo, permitiendo la evolución del sistema y detectando con antelación los errores;
e. Creer en la comunicación oral, la colaboración estrecha de programadores, las pruebas y el código fuente, para definir la estructura del sistema y los objetivos. Confiar en el proceso de evolución del proyecto, que dura tanto como el sistema;
f. Creer en prácticas de corto plazo que trabajan con las aptitudes de los programadores y, los intereses a largo plazo del proyecto.

A. Los Cuatro Valores del XP

XP utiliza un conjunto de valores, principios y reglas básicas que pretenden alcanzar eficiencia y efectividad en el proceso de desarrollo de software (Beck, 1999). Los valores son cuatro; comunicación, simplicidad, retroalimentación y coraje. En estos valores, están fundamentados algunos principios básicos; retroalimentación rápida, simplicidad, cambios incrementales, comprensión a los cambios y calidad del trabajo. Usando estos principios, fueron definidas las doce prácticas básicas que son adoptadas por el XP. Los cuatro valores del XP son:

a. Comunicación.- se debe preferir chat a correo, conversar personalmente a llamadas, trabajar en la misma sala a tener salas aisladas, trabajar en conjunto para revisar el resultado final. Jeffries (2001) resalta que debe existir comunicación en todos los sentidos. La comunicación debe ser fluida en todo el equipo de XP;
b. Simplicidad.- el objetivo es simplificar continuamente el software, esto

sostiene la premisa de extremo, porque la simplicidad no es fácil. Simplicidad y comunicación están directamente relacionadas, porque cuando existe más comunicación en el equipo, más claro esta lo que debe hacerse y, más seguro lo que no debe hacerse. Entonces, el software será más simple;

c. Retroalimentación.- todo problema debe ser descubierto lo más pronto posible para que pueda ser corregido lo más rápido posible. Toda oportunidad debe ser descubierta lo más pronto posible para que pueda ser aprovechada lo más rápido posible figura 2.20;

d. Coraje.- Este valor esta relacionado con los tres primeros valores (Beck, 1999). Es preciso el coraje para identificar un problema en el proyecto, pedir ayuda cuando sea necesario, simplificar el código que está funcionando, informar al cliente sobre plazos estimados que no será suficiente, hacer alteraciones durante el desarrollo.

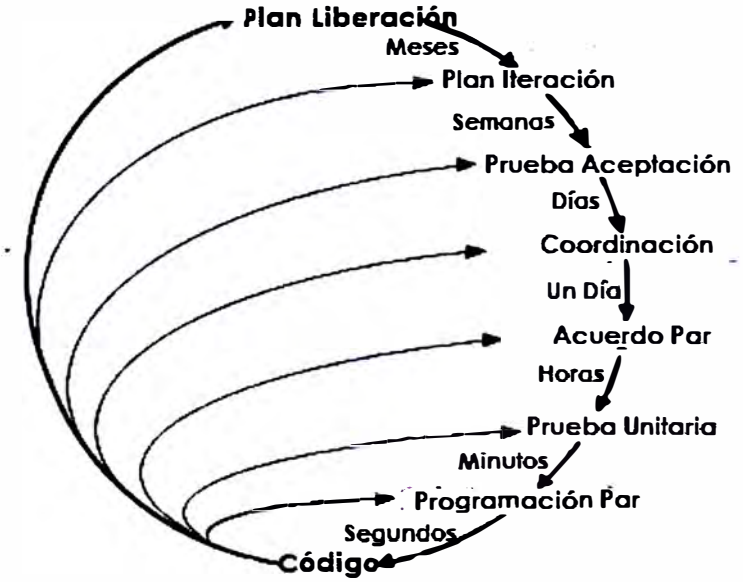

Figura N° 2.20: XP - Planificación y lazos de retroalimentación. (Wells, 2001)

B. Las Doce Prácticas de XP

Cada práctica esta relacionada con otra, permitiendo que las debilidades de una práctica sean mejoradas con las fortalezas de las otras prácticas. (Beck, 1999)

La figura 2.21, representa la colaboración entre las prácticas, cada práctica es

un componente simple, la riqueza radica en la interacción de las prácticas. La línea de conexión entre dos prácticas significa que ellas colaboran una con otra.

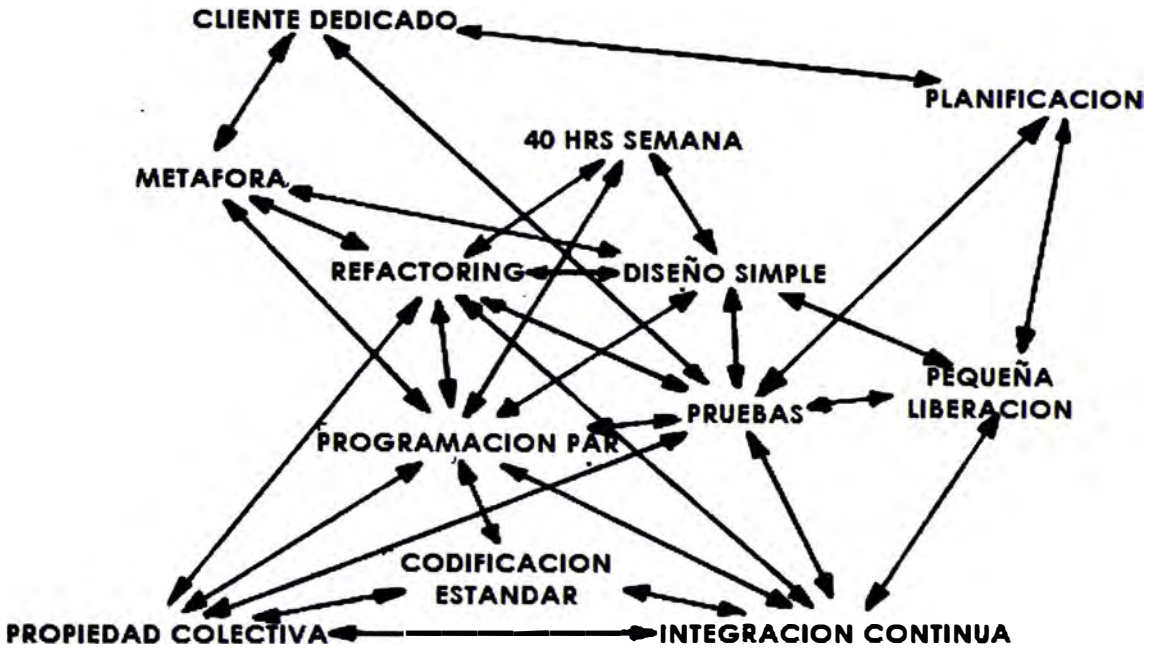

Figura N° 2.21: XP - Colaboración entre las prácticas. (Beck, 1999)

B.1 Juego de Planificación (Planning Game)

Conforme a Beck (1999), práctica compuesta por decisiones del negocio (cliente) y decisiones técnicas (programadores y otros). El cliente decide sobre: cuando esta listo el producto, la prioridad, composición de las versiones y fechas de entregas de versiones. Las decisiones técnicas son: tiempo para implementar una historia de usuario, riesgos técnicos sobre decisiones del negocio, organización del proceso y equipo, planificación detallada (que historia de usuario se hace primero). Como práctica, estructura el plan y actualiza el plan.

La planificación en XP, responde dos preguntas claves en el desarrollo de software: predecir que hacer hasta determinada fecha y, determinar cual es la próxima tarea. Énfasis en guiar el proyecto, en lugar de especificar exactamente lo que se necesita y cuánto tiempo implica, lo cual es bastante difícil (Jeffries, 2001). Existen dos pasos en la planificación de XP, siendo:

Planificación de versión (Release Planning), paso donde el cliente presenta las características deseadas del software a los programadores y ellos estiman su dificultad. XP trabaja sobre versiones frecuentes y pequeñas cada dos o tres meses, estimando las dificultades y conociendo las características, el cliente alcanza un plan del proyecto, al inicio los planes de las versiones son inexactos, hasta que el equipo comienza a trabajar. Entonces, podemos saber en que tiempo se ejecutara el trabajo. La tabla 2.14 muestra el modelo de una tarjeta de historia de usuario.

Historia de Usuario	
Número:	Usuario:
Nombre de historia:	
Prioridad en negocio: (Alta / Media / Baja)	Riesgo en desarrollo: (Alta / Media / Baja)
Esfuerzo:	Iteración asignada:
Programador responsable:	
Descripción:	
Observaciones:	

Tabla N° 2.14: Tarjeta historia de usuario. (Universidad Politécnica de Valencia, 2006)

Planificación de la iteración (Iteration Planning), paso donde el equipo recibe orientación mediante tarjetas de tareas (Beck, 1999), la tabla 2.15 muestra un modelo de tarjeta de tarea. Los equipos XP construyen software en "iteraciones" de dos a tres semanas, entregando un software ejecutable y útil al final de cada iteración. Durante la planificación de versión, el cliente presenta tarjetas de usuario para las próximas dos o tres semanas. Los programadores parten las historias de usuario en tareas, y estiman el tiempo (más detallado que en la planificación de versión). El equipo en base a la iteración previa, estima el tiempo necesario para la siguiente iteración.

Usando este enfoque, cada dos o tres semanas debe verse el progreso, los requerimientos deben ser cubiertos totalmente. Por un lado, el cliente puede cancelar el proyecto si el progreso no adecuado, por otro lado, si el progreso es visible, decidir que hacer la próxima iteración. (Jeffries, 2001)

Tarea de Ingeniería	
Número tarea:	Número historia:
Nombre tarea:	
Tipo de tarea:	Esfuerzo:
Fecha inicio:	Fecha fin:
Programador responsable:	
Descripción:	

Tabla N° 2.15: Tarjeta de tarea de ingeniería. (Universidad Politécnica de Valencia, 2006)

B.2 Pequeñas Versiones (Short Releases)

El equipo XP coloca rápidamente un pequeño sistema en producción, y lo actualiza mediante ciclos bastante cortos. Según Jeffries (2001), los equipos XP trabajan con pequeñas versiones de dos modos importantes:

a. Primero, el equipo pone a disposición del cliente una versión de software ejecutable y probado, con características escogidas por el cliente para la iteración. El cliente usa este software, sea para evaluar, o para entregar a los usuarios finales. El aspecto importante es que el software es visible y, entregado al cliente al final de cada iteración.
b. Segundo, el equipo XP entrega a los usuarios finales una versión de proyectos Web XP con duración mensual, las iteraciones más pequeñas son puestas a disposición diariamente cuando sea posible.

Todas las versiones deben ser tan pequeñas cuánto sea posible y, deben contener los requisitos más importantes. (Beck, 1999)

B.3 Metáfora (Methapor)

Guían todo el desarrollo y la comunicación con el cliente utilizando un "sistema de nombres" y una descripción de la arquitectura técnica.

Un proyecto de software en XP esta guiado por una metáfora simple (Beck, 2002). La metáfora mantiene a todos los desarrolladores en sintonía con el proyecto y ayuda a definir nombres de objetos o clases. Es importante como práctica para mantener el código estándar.

Jeffries (2001) opina que una metáfora poética no sirve. Los equipos XP usan

un sistema común de nombres para que todos entiendan como trabaja el sistema. Así, podemos administrar mejor las funcionalidades.

B.4 Diseño Simple (Simple Design)

Los equipos XP construyen el software de acuerdo al diseño más simple y satisfaciendo los requisitos, manteniendo el proyecto alineado a la funcionalidad actual del sistema. Así, no perdemos nada, el software está siempre para la siguiente iteración. (Jeffries, 2001)

Beck (1999) afirma que el diseño correcto para un software en cualquier momento es aquel que:

a. Supera todas las pruebas;
b. No presenta lógica duplicada (cuidar la jerarquía de clases paralelas);
c. Expone claramente la intención de los programadores;
d. Tiene el mínimo número de clases y métodos.

El diseño en XP no se hace una sola vez, se hace todo el tiempo, porque existen pasos del diseño realizados en la planificación de versión y planificación de iteración, el equipo participa en esas sesiones y en la revisión rápida del diseño por refactoring. En procesos incrementales e iterativos, como extreme programming, un buen diseño es indispensable. Por eso existe el enfoque del diseño a lo largo de todo el desarrollo. (Jeffries, 2001)

B.5 Pruebas (Testing)

Los equipos XP validan el software durante todo el proceso. Según Beck (2002) los programadores escriben primero las pruebas unitarias y continúan el desarrollo que satisface estas pruebas. Los clientes escriben las pruebas de aceptación para validar si los requisitos están siendo atendidos.

Fowler y Foemmel (2006) enfatizan que las pruebas unitarias escritos por los programadores tienen por finalidad probar una clase individual o un grupo de clases. Las pruebas funcionales (pruebas de aceptación) son escritos por el cliente y tienen la finalidad de probar el sistema de inicio-a-fin. Es importante resaltar que las pruebas son escritas antes de construir el código.

De acuerdo a Wells (2006), las pruebas unitarias son elementos clave en XP, creadas antes de codificar y almacenados en un repositorio junto al código que será probado. Las pruebas unitarias deben ser automatizadas, por el menor costo que puede proporcionar al identificar errores. Entonces, un conjunto completo de pruebas unitarias deben estar disponibles al inicio del proyecto, no al final.

Las pruebas de aceptación constituyen la primera indicación que las historias de los clientes son válidas, muestran que cada historia puede ser probada. Normalmente, son escritos por los clientes o usuarios mediante las tarjetas de historia Wells (2006), con ayuda de un miembro del equipo responsable de probar el sistema. Las pruebas de aceptación también son usadas como pruebas de validación, antes de liberar una versión del sistema.

B.6 Refactoring

El equipo busca perfeccionar el diseño del sistema durante todo el desarrollo, manteniendo la claridad del software: sin ambigüedad, con alta comunicación, simple y completo.

XP usa un proceso continuo de mejora de diseño llamado refactoring, del título del libro de Martin Fowler, "Refactoring: Improving the Design of Existing Code" (Jeffries, 2001).

Refactoring es la técnica empleada para reestructurar el código de software que esta funcionando, cuya principal finalidad es hacer que un programa quede más reutilizable y fácil de mantener, sin que haya cambio en su comportamiento (Fowler, 2005).

El objetivo del refactoring es remover código duplicado (una señal de diseño pobre) y, añadir "cohesión" al código, mientras baja el acoplamiento, "alta cohesión y bajo acoplamiento" es indicador de código correcto. Los equipos XP comienzan con una ventaja, diseño simple, y continúan con esa ventaja durante todo el proceso. Fowler (2005) afirma que un buen diseño es esencial para mantener la velocidad del desarrollo del software. Entonces, la técnica

de refactoring ayuda a desarrollar el código más rápidamente.

B.7 Programación por Pares (Pair Programming)

Según Jeffries (2001), "Todo software producido con XP es construido por dos programadores, sentados lado a lado, en la misma máquina. Esta práctica asegura que todo código producido es revisado por el otro programador". El resultado de esta práctica es un diseño rápido, con pruebas más eficaces y mejor código.

Existen dos roles para cada programador, el que esta con el teclado y mouse, piensa la forma de implementar el método. Mientras que el otro piensa si aquel enfoque funcionara, si existe una prueba no realizada y si existe una forma simple para construir el sistema y desaparecer el problema (Beck, 1999).

Para Jeffries (2001), el noventa por ciento de los programadores que aprenden programación por pares la prefieren. XP recomienda la programación en pares para todos los equipos, técnica que además de proveer código y mejores pruebas, también sirve para transferir conocimiento a todo el equipo.

B.8 Propiedad Colectiva (Colletive Ownership)

En XP, cualquier par de programadores puede mejorar cualquier código en cualquier momento. Entonces, todo código tiene mucha atención de los programadores, añadiendo calidad al código y reduciendo defectos. (Jeffries, 2001)

De acuerdo a Beck (1999), el equipo XP es responsable del código, los programadores no necesitan pedir autorización para modificar cualquier programa. Esto es posible con pruebas unitarias correctamente elaboradas, que dan seguridad a los programadores. La práctica de integración continua apoya la práctica de propiedad colectiva, donde los programadores ni perciben, si el código que ellos programaron fue mejorado o ampliado.

B.9 Integración Contínua (Continuous Integration)

Los equipos XP mantienen el sistema integrado todo el tiempo, el

código es integrado y probado luego de algunas horas, a lo sumo tras un día de desarrollo, para realizar esto, tener una máquina dedicada para la integración (Beck, 1999). Contar con una herramienta para controlar la versión.

Según Wells (2006), la integración contínua evita la dispersión de esfuerzos de desarrollo, los programadores deben estar en constante comunicación, para saber que reutilizar o que compartir.

El equipo XP trabaja con la versión más reciente, la integración debe realizarse varias veces por día, cada par de programadores es responsable por integrar su propio código. Esto se realiza cuando las pruebas unitarias se han ejecutado al 100% para cada funcionalidad planeada.

B.10 Semana de 40 horas (40 Hour Week)

Como regla, no se debe trabajar más de cuarenta horas por semana, los programadores exhaustos cometen más errores.

Para Beck (1999), la sobrecarga de trabajo es síntoma de serios problemas en el proyecto, la regla es simple, no trabajar una segunda semana haciendo horas extras. Si una semana se trabaja horas de más, entonces la planificación esta en desorden. Si se repite la semana siguiente, indica que algo está mal.

El equipo XP debe trabajar intensamente para maximizar la productividad, un programador cansado, raciocina lentamente y esta más distraído. Un programador cansado inserta errores en un programa, errores que darán trabajo al corregirlos, consumiendo más horas extras.

B.11 Cliente Dedicado (On-Site Customer)

El equipo XP tiene el "cliente real" (usará el sistema) disponible todo el tiempo, que determina los requisitos, asigna prioridades y, responde las dudas.

Beck (1999) define al "cliente real" como, aquel cliente que utilizará el sistema cuando esté en producción. La regla persigue tener el usuario real del sistema, de preferencia junto al equipo de desarrollado.

Si el cliente no esta en el ambiente de los programadores, todavía es posible trabajar con XP, debemos tener un canal de comunicación abierto (Internet) y que el cliente tenga disponibilidad para:

a. Atender a los programadores cuando surgen dudas;
b. Producir valor para el diseño escribiendo pruebas funcionales;
c. Definir las prioridades para los programadores.

Si el equipo XP no puede incluir un cliente real, debe adicionar un riesgo al proyecto, porque los programadores codificaran sin saber exactamente que pruebas deben satisfacer.

B.12 Estándares de Programación (Coding Standards)

Los programadores deben escriben código con el mismo estilo, homogéneo y legible de acuerdo a reglas establecidas. El equipo XP sigue un estándar de codificación para todos sus miembros, así, el código del sistema podrá verse como si fuera escrito por un solo programador. Los aspectos particulares del estándar no importan, lo importante es que todo el código debe ser uniforme, en defensa de la propiedad colectiva. (Jeffries, 2001)

Según Wells (2006), la estandarización del código lo mantiene consistente, fácil de entender y para hacer refactoring por el equipo. No importa cual es el estándar, suficiente que haya uno. El estándar no necesita ser definido específicamente, puede emerger naturalmente como resultado de la propiedad colectiva (programación en pares).

Las prácticas se agrupan en:

a. Retroalimentación a escala fina: desarrollo guiado por pruebas, juego de planificación, cliente dedicado, programación por pares;
b. Proceso contínuo en lugar de por lotes: integración continua, refactoring sin piedad, pequeñas versiones;
c. Entendimiento compartido: diseño simple, metáfora, propiedad colectiva del código, estándares de programación;
d. Bienestar del programador, semana de 40 horas.

C. Ciclo de Vida y Fases del Proceso XP

El proyecto ideal con XP comienza con una fase corta de desarrollo, seguida por años de producción, refinamientos simultáneos y finalmente concluye cuando el proyecto no tiene más sentido. (Beck, 1999)

Según Wake (2002), el ciclo de vida de extreme programming es: análisis, prueba, codificación y diseño (en este orden) con realimentación hacia análisis. El ciclo de vida de XP es bastante corto, y diferente a los modelos de procesos convencionales, ver figura 2.22.

C.1 Exploración

Para Wake (2002), en la fase de exploración se entiende que debe hacer el sistema, el cliente escribe las historias y el programador estima las historias, concluye si todas las historias están estimadas y registradas tabla 2.16 para la próxima fase (planificación), ver figura 2.22 y 2.23.

En la fase de exploración con las tecnologías exploradas debe comenzar y finalizar el desarrollo de un programa. El equipo debe conocer las tecnologías que esta explorando para el proyecto. (Beck, 1999)

Nro. HU	Historia de Usuario (HU)	Esfuerzo (semana)

Tabla N° 2.16: Formato para plan de alto nivel

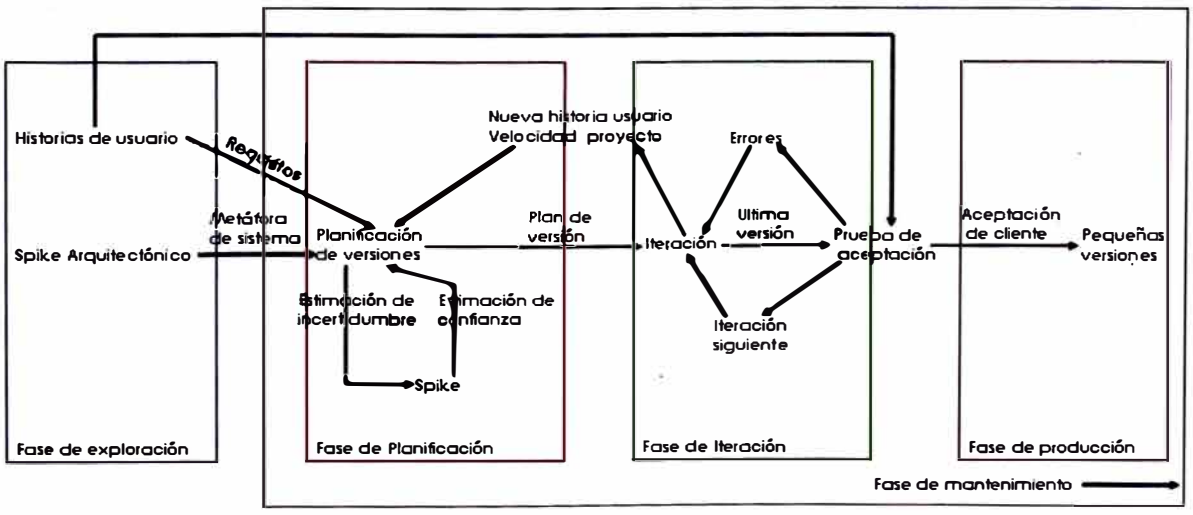

Figura N° 2.22: XP - Ciclo de Vida, adaptado. (Beck, 1999)

Wake (2002) afirma que las historias pequeñas tienen menor riesgo, si una historia es muy grande el cliente debe partir la historia, porque los programadores tienen dificultades para estimar. Si los programadores no pueden estimar, ellos deben hacer una programación rápida de la historia, ósea una puntuación (spike), con duración de minutos, horas o dos días.

En paraleló a las historias (clientes), los programadores están probando la arquitectura del sistema (técnica), esta tarea consume una o dos semanas, deben probar de tres o cuatro forma diferentes, diferentes pares prueban diferentes tecnologías y comparan, si el tiempo no es suficiente para probar la tecnología se clasifica como un riesgo para el proyecto. Durante la fase de exploración se puede invitar a un especialista en la tecnología seleccionada, quien no perderá tiempo en problemas, porque sabe por experiencia como resolverlos.

Beck (1999) destaca que la exploración de la arquitectura técnica, ayuda a los programadores a tener una idea de la implementación cuando el cliente presenta sus historias de usuario, los programadores necesitan estimar cada tarea de programación durante la exploración. La práctica de estimar crea confianza en el equipo para cuando ellos tengan que estimar el tiempo.

C.2 Planificación

El objetivo es definir la fecha más próxima y el mayor conjunto de historias de usuario que serán concluidas en la primera versión, definición de acuerdo a estimaciones entre cliente y programadores. (Beck, 1999)

Wake (2002), nos presenta pasos, figura 2.23, para la fase de planificación de versión, siendo:

a. El cliente selecciona las historias por prioridad: alto, medio o bajo;
b. Los programadores califican las historias por riesgo: alto, medio o bajo;
c. Los programadores asignan la velocidad, calculada mediante la estimación de las historias en la fase de exploración. La velocidad es fijada empíricamente, por la experiencia de los programadores;
d. Los clientes escogen las historias para la próxima versión.

El plan de versión es registrado de acuerdo en la tabla 2.17, como sigue.

N°	Historia de Usuario	Prioridad	Riesgo	Esfuerzo(semana)	Iteración

Tabla N° 2.17: Formato para plan de versión

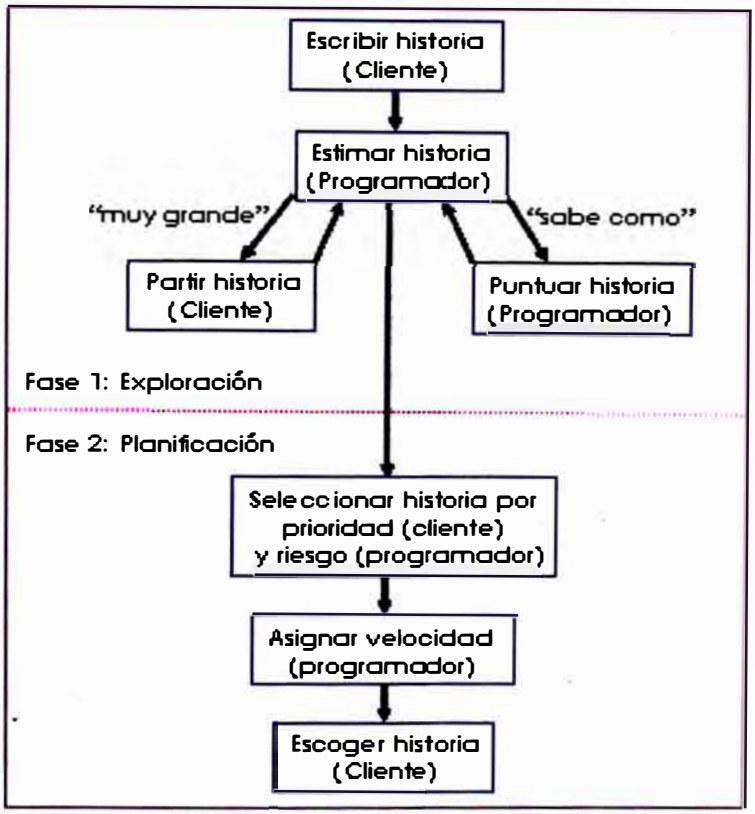

Figura N° 2.23: XP - Planificación de versión. (Wake, 2002)

Las historias son recolectadas, el cliente decide que historias son prioritarias y deben formar parte de cada versión, hacer una lista priorizada de las historias, los programadores asignan velocidad a cada historia de usuario (tiempo de implementación). Si la fase de exploración es correcta, se necesita sólo algunos días para hacer la fase de planificación de versión.

C.3 Iteración

Para Wake (2002), las historias de usuario de los clientes son partidas en tareas de ingeniería (task cards), definiendo que programadores trabajan en cada tarea. El primer día de cada iteración, el equipo decide que historias serán trabajadas, los programadores seleccionan las tareas, estiman (en días)

y aceptan las tareas que programaran. La figura 2.24 muestra como funciona la planificación de iteración y la tabla 2.18 el registro del plan de iteración.

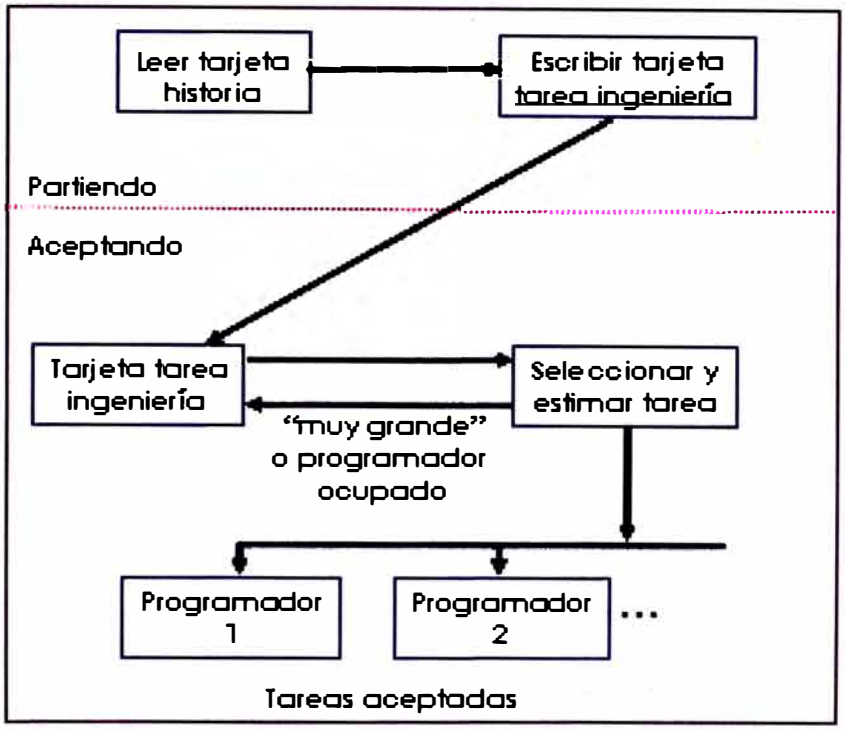

Figura N° 2.24: XP - Planificación de Iteración. (Wake, 2002)

El plan de versión contiene iteraciones, para cada iteración están las historias de usuario priorizadas por el cliente, agrupadas por la misma prioridad. Sin embargo, el negocio debe conocer las liberaciones, que son las historias de usuario implementadas para una iteración. (Beck, 1999)

Según Beck (1999) cada iteración produce un conjunto de casos de pruebas de aceptación para cada historia calendarizada para una iteración que dura de una a cuatro semanas. La primera iteración, muestra el comportamiento de la arquitectura del sistema. Las historias deben escogerse de forma que sean el eje para crear todo el sistema, la pregunta clave en esta fase es, ¿Qué cosa de mayor valor debe trabajar el equipo en esta iteración?.

Una característica importante de XP es, implementar una funcionalidad seleccionada para la iteración en curso (Wells, 2006). Por esto, habrá tiempo para implementar otra funcionalidad en una próxima iteración, porque la historia del cliente esta seleccionada como la siguiente en importancia en la

planificación de versión. Entonces, el tiempo de cada iteración es respetado y se tiene la velocidad real del proyecto durante la iteración, por ello, el conjunto de historias de la nueva iteración estará correctamente estimado.

Nº HU	Nº TI	Fecha Inicio	Fecha Fin	Programador	Incidencia

Tabla Nº 2.18: Plan de iteración

Al finalizar cada iteración el cliente habrá completado la ejecución de todas las pruebas funcionales y, al final de la última iteración, el sistema estará completo para entrar en producción.

C.4 Producción

La producción del sistema puede comenzar cuando un ciclo de realimentación ha concluido, al final de una versión, como se muestra en la figura 2.22.

Según Beck (1999), existen algunas pruebas aplicadas (como estabilidad del sistema) en esta fase, para evaluar si el software realmente está para entrar en producción.

Puede ser necesario ajustar la eficiencia u otros, el mejor momento para hacerlo es cuando se concluye la versión, estas pruebas serán realizadas en la máquina de producción.

C.5 Mantenimiento

La fase de mantenimiento es el estado normal de un proyecto XP, ver figura 2.22. Se debe simultáneamente producir nuevas funcionalidades, mantener el sistema existente funcionando, sustituir miembros del equipo que se retiran e incorporar nuevos miembros. (Beck, 1999)

En esta fase, se puede intentar hacer refactoring mayor. Se puede probar nuevas tecnologías que se pretende incorporar en las próximas versiones, o migrar la tecnología que está en uso hacia versiones más actualizadas. El cliente puede escribir nuevas historias que apoyan mejor a su negocio.

Cuando el sistema está en producción, es probable que la velocidad de desarrollo cambie. Beck (2000) cree que hay un aumento en 50% del tiempo requerido antes de la producción (estimado de 2 a 3 días). Sin embargo, no se debe adivinar, se debe estimar.

Se debe tener cuidado con la rotación de los programadores, porque hay cosas que se aprenden dando soporte a la producción y no se aprende de otra forma. Para disminuir riesgos, el equipo debe ser cambiado gradualmente.

C.6 Fin del Proyecto

Beck (1999) considera que "Morir bien es tan importante como vivir bien. Una verdad para XP como para las personas". Cuando no hay nuevas historias, es el momento de finalizar el proyecto, momento para escribir algunas páginas (5 a 10) sobre la funcionalidad del sistema, un documento que ayude en el futuro ha realizar el mantenimiento.

D. Roles del Equipo

Algunos roles esenciales deben existir en un equipo XP: programador, cliente, entrenador y supervisor. (Beck, 1999)

Programador (programmer) es el corazón de XP, es responsable:

a. Definir la arquitectura técnica;
b. Estimar los plazos para las historias de usuario del cliente (estimar);
c. Solicitar al cliente que aclare o divida una historia de usuario cuando es necesario (decisión técnica).
d. Definir las tareas de ingeniería desde las historias de usuario (decisión técnica);
e. Estimar el tiempo para desarrollar las tarea de ingeniería (estimar);
f. Integridad del sistema (pruebas unitarias y de integración);
g. Del diseño (refactorización, simplicidad);
h. La construcción del código (codificar);
i. Trabajar en par (capacidad de comunicación);
j. Aceptar críticas (código colectivo).

Cliente (customer) otro rol importante en XP, el programador sabe como programar, el cliente sabe que se debe programar, responsable de:

a. Proporcionar información sobre características del negocio;
b. Definir los requisitos del software;
c. Escribir las historias de usuario;
d. Definir las prioridades para las historias de usuario;
e. Especificar las pruebas de aceptación;
f. Esclarecer dudas siempre que sea solicitado.

Encargado de pruebas (tester), en XP su rol esta enfocado al cliente, tiene por responsabilidad:

a. Definir con el cliente las pruebas de aceptación;
b. Apoyar al cliente en la preparación de las pruebas de aceptación;
c. Ejecutar las pruebas y publicar los resultados para el equipo;
d. Asegurarse que las pruebas de aceptación se han superado.

Supervisor (tracker) es la conciencia del equipo, sus responsabilidades son:

a. Recoger, analizar y publicar información sobre el desarrollo del proyecto sin afectarlo, para conocimiento del equipo;
b. Supervisar el cumplimiento de las estimaciones para cada iteración;
c. Informar el progreso de la iteración en curso;
d. Recolectar datos históricos sobre signos del proyecto (métricas);
e. Tomar decisiones siempre que las cosas parecen ir lento.

Entrenador (coach), es un experto en XP, tiene las funciones siguientes:

a. Responsable del proceso en conjunto;
b. Guiar al grupo de forma indirecta e intervenir si es necesario;
c. Identificar las desviaciones y reclamar su atención a las personas que se desvían del proceso;
d. Mantener la visión del proyecto;
e. Comunicar una tarea a un programador para su desarrollo.

Jefe de proyecto (manager), persona que transmite coraje, confianza y establece responsabilidades de cada uno, tiene varias funciones, como:

a. Gerenciar el equipo y los problemas del equipo;
b. Organizar y guiar las reuniones de planificación;
c. Facilitar la relación entre clientes y desarrolladores;
d. Registrar lo tratado y definido en las reuniones;
e. Mantener al supervisor (tracker) informado de las reuniones;
f. Buscar recursos.

2.2.4 CALIDAD DEL SOFTWARE

Calidad

"La calidad es cumplir con los requerimientos del usuario". (Crosby, 1979)

"La calidad es el grado en el que un conjunto de características inherentes cumple con los requisitos". (ISO 9000: 2000)

"La calidad tiene que ser construida en cada diseño y cada proceso. No puede ser creada por medio de la inspección". (Ishikawa, 1986)

Los conceptos mencionados interpretados en el contexto de la calidad del software, indican que la calidad del software es cumplir con los requerimientos del usuario y, ser construida en cada diseño y cada proceso.

Calidad de Software

La calidad de software, es la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer sus necesidades explícitas o implícitas. (ISO/IEC 9126-1: 2001)

Durante el desarrollo del software, la calidad comprende los requisitos, especificaciones y diseño del sistema. La calidad de concordancia es un aspecto centrado en la implementación. (Pressman, 2002; ápud Glass, 1998), establece una relación más "intuitiva":

> *Satisfacción del usuario = producto satisfactorio + buena calidad + entrega dentro de presupuesto y del tiempo establecidos.*

"La concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente". (Pressman, 2002)

Las necesidades explícitas son los requisitos y características transmitidos por el usuario al desarrollador, las necesidades implícitas no se formalizan, pueden ser facilidad de uso, velocidad de ejecución, etc. de los productos de software.

Por otra parte, el control de calidad de los productos software se enfoca en los entregables con defectos terminados para corregirlos y, el aseguramiento de la calidad, utiliza los resultados del control de calidad para evaluar y mejorar los procesos que usamos para desarrollar el producto software.

La investigación guiada por los objetivos como: determinar las características y los atributos de calidad del producto software, elaborar los procesos de desarrollo de software aplicando las metodologías XP y ICONIX, comparar los dos procesos de desarrollo de software mediante los atributos de calidad del producto software, para las características de funcionalidad, fiabilidad, usabilidad, eficiencia, facilidad de mantenimiento y portabilidad. Por estas consideraciones, los aspectos de calidad de software serán tratados de acuerdo a las normas técnicas peruanas NTP ISO/IEC 9126-1:2004, NTP ISO/IEC 9126-2:2004, NTP ISO/IEC 9126-3:2005, NTP ISO/IEC 14598-1:2005 y NTP ISO 14598-3:2005, para evaluar la calidad del producto software. Por tanto, la revisión literaria sobre la calidad de producto software, esta referido a dichas normas, excepto los casos en que se referencia explícitamente.

2.2.4.1 Marco de Trabajo del Modelo de Calidad

En general un modelo de calidad, descompone la calidad jerárquicamente en un conjunto de características y sub características, que pueden usarse como una lista para evaluar la calidad del software.

A. Ciclo de Vida de Calidad del Software

Los usuarios plantean sus requerimientos de calidad del software, estos

requerimientos son usados cuando se especifican la calidad externa e interna mediante características y sub características. La calidad del producto software puede evaluarse midiendo los atributos internos (medidas estáticas de productos intermedios), o midiendo los atributos externos (midiendo el comportamiento del código cuando es ejecutado) o midiendo los atributos de calidad en uso. La figura 2.25 muestra que la calidad del proceso contribuye a mejorar la calidad del producto y este a la calidad en uso. En forma inversa, evaluando la calidad en uso podemos retroalimentar para mejorar la calidad del producto y evaluando el producto podemos retroalimentar para mejorar los procesos. (NTP-ISO/IEC 9126-1:2004)

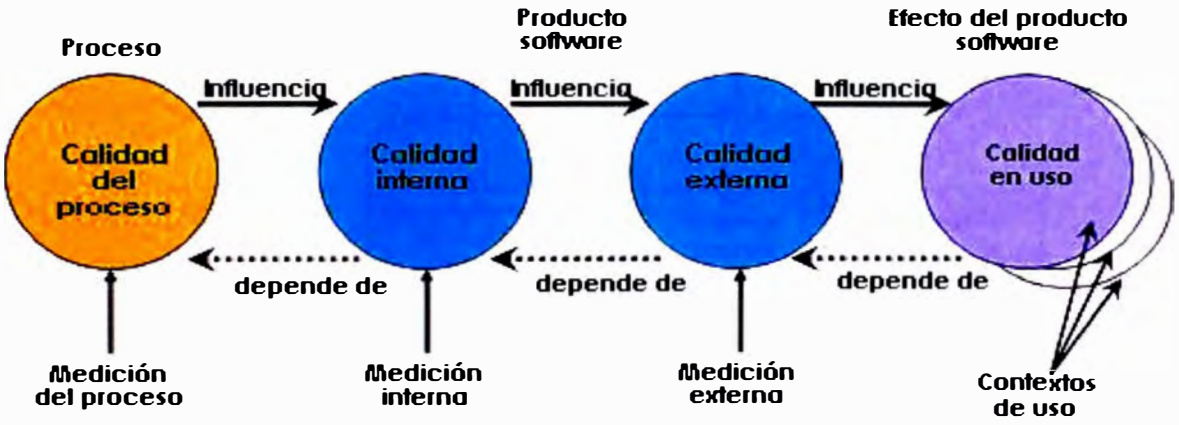

Figura N° 2.25: Ciclo de vida de la calidad del software. (NTP-ISO/IEC 9126-1:2004)

B. La Calidad en el Ciclo de Vida del Software

Durante la formulación de requisitos y análisis, la calidad es especificada por los requisitos de los usuarios. En las fases de diseño e implementación, la calidad externa se traduce en un diseño técnico. No existe calidad perfecta o absoluta, solamente existe una calidad necesaria y suficiente que satisface las necesidades reales de los usuarios, cuando el producto es entregado y utilizado.

Necesidades de calidad del usuario.- son definidas como requerimientos de calidad por métricas de calidad en uso, métricas externas o métricas internas. Requerimientos usados como criterios cuando un producto es validado. Un producto satisface las necesidades del usuario cuando el desarrollo del software es iterativo y realimentación continua desde la visión del usuario.

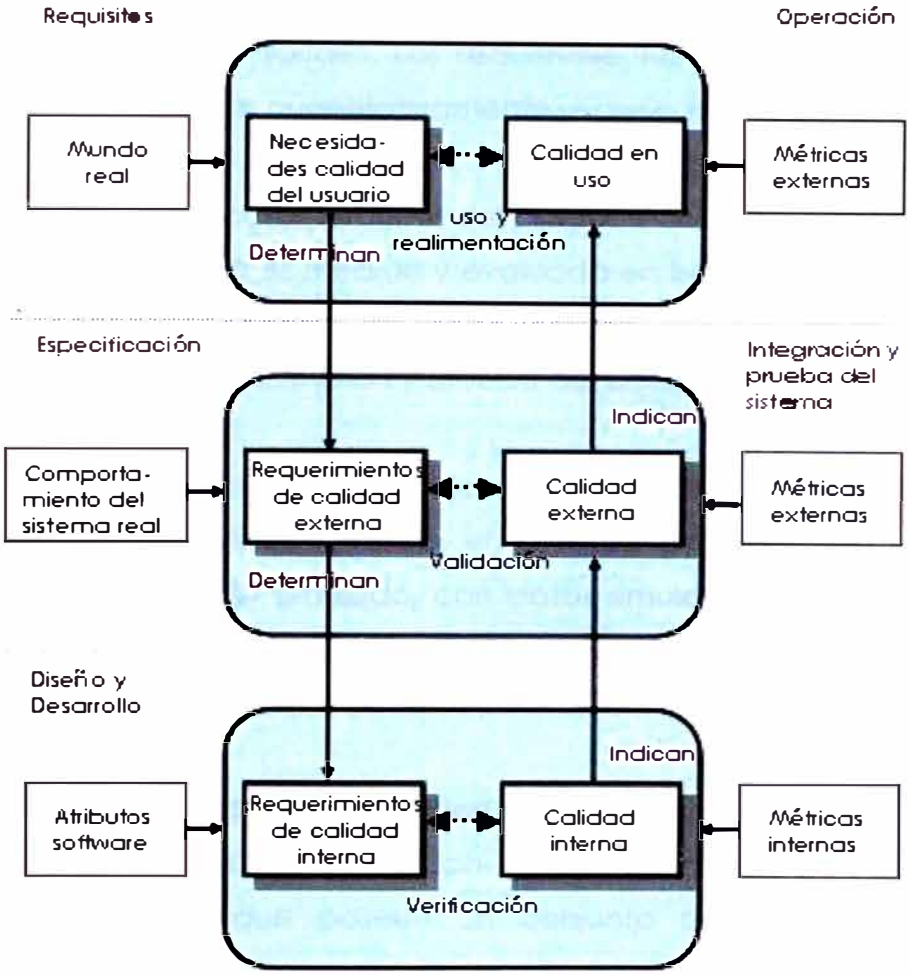

Figura N° 2.26: Calidad en el ciclo de vida del software. (NTP ISO/IEC 14598-1:2005)

Requerimientos de calidad externa.- incluyen requerimientos derivados de las necesidades de calidad de usuarios y, calidad en requerimientos de uso. Estos requerimientos son usados como los objetivos para la validación en varias etapas de desarrollo. Los requerimientos de calidad externos para las características de calidad de NTP ISO/IEC 9126-1:2004 deben definirse como requerimientos de calidad usando métricas externas, deben transformarse en requerimientos de calidad internos y deben usarse como criterios para evaluar un producto.

Requerimientos de calidad interna.- son usados para definir propiedades internas de productos. Estos pueden incluir modelos estáticos y dinámicos, otros documentos y código fuente. Estos requerimientos son usados como objetivos para la validación en varias etapas de desarrollo, también pueden

ser usados para definir estrategias de desarrollo, criterios de evaluación y verificación durante el desarrollo. Los requerimientos específicos de calidad interna deben ser descritos cuantitativamente usando métricas internas.

Calidad interna.- son las características del producto software desde una visión interna. La calidad interna es medida y evaluada en base a los requerimientos de calidad interna. La calidad del producto software puede ser mejorada durante la implementación, revisión y prueba del código fuente.

Calidad externa.- son las características del producto software desde una visión externa. Es la calidad cuando el software es ejecutado, medida y evaluada en un ambiente simulado, con datos simulados y usando métricas externas. Durante las pruebas, muchas fallas son descubiertas y eliminadas. Sin embargo, algunas fallas todavía pueden permanecer después de las pruebas.

2.2.4.2 Modelo de Calidad para la Calidad Externa e Interna

Según Scalone (2006), la calidad a nivel producto, plantea distintos modelos y estándares que poseen un conjunto de características, sub características, atributos y métricas. El equipo de desarrollo debe evaluar la calidad del software durante el ciclo de vida del software.

Nivel de Calidad	Modelo de Calidad de Software	Estándar de Calidad de Software
Proceso	CMMi TickIT Bootstrap Personal Sw Process (PSP) Team Sw Process (TSP) Practical Sw Measurement (PSM) Six Sigma for Software	ISO 90003 ISO 12207 ISO 15504 (SPICE) IEEE / EIA 12207 ISO 20000 ITIL Cobit 4.0
Producto	Gilb GQM Mc Call Furps Boehm SATC Dromey C-QM Metodología SQAE WebEQM	ISO/IEC 9126-1 ISO 25000 (SQUARE) IEEE Std 1061-1998

Tabla N° 2.19: Modelos y estándares para calidad de software. (Scalone, 2006)

Modelo de Gib	Modelo de McCall	Modelo de Boehm	Estándar ISO/IEC 9126
• Corrección • Facilidad de mantenimiento • Integridad • Facilidad de uso	• Corrección • Fiabilidad • Eficiencia • Integridad • Facilidad de uso • Mantenibilidad • Flexibilidad • Facilidad de prueba • Portabilidad • Reusabilidad • Interoperabilidad	• Usabilidad • Claridad • Eficiencia • Fiabilidad • Modificabilidad • Reusabilidad • Modularidad • Documentación • Elasticidad • Corrección • Mantenibilidad • Portabilidad • Interoperabilidad • Entendebilidad • Integridad • Validabilidad • Flexibilidad • Generalidad • Economía	• Funcionalidad • fiabilidad • Usabilidad • Eficiencia • Mantenibilidad • Portabilidad

Tabla N° 2.20: Características de calidad de producto software.

Para cumplir con la conformidad de cada característica, usaremos las Normas Técnicas Peruana (NTP) y, los indicadores de la variable independiente de investigación, la NTP ISO/IEC 9126-1:2004 para el modelo de calidad externa e interna ver anexo B y figura 2.27. Las sub características se medirán por métricas internas y externas de acuerdo a NTP ISO/IEC 9126-2:2004 y NTP ISO/IEC 9126-3:2005 ver anexo C.

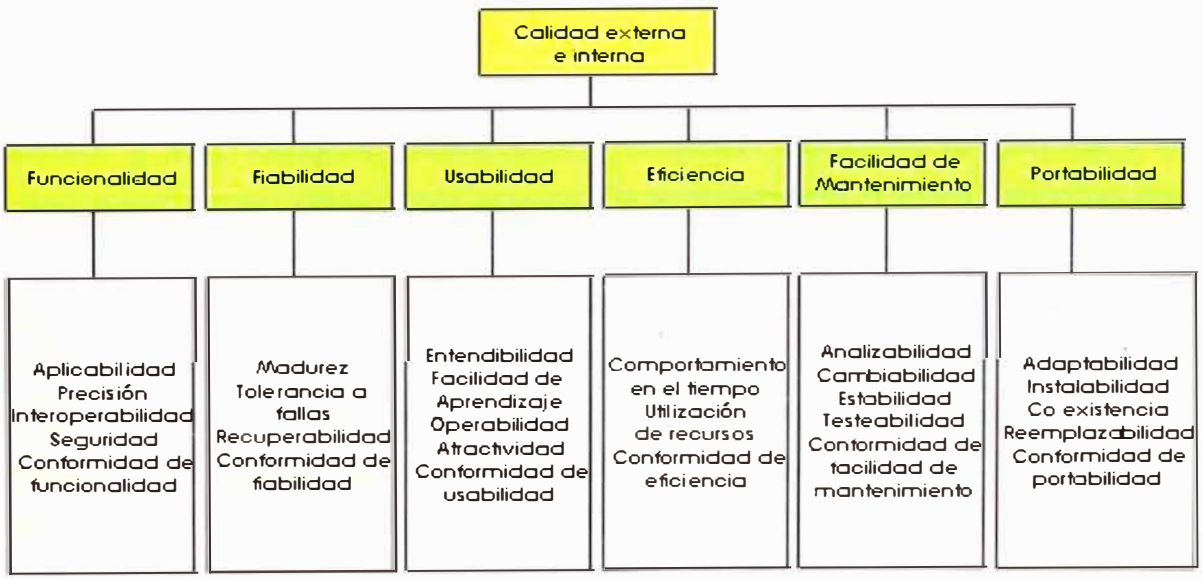

Figura N° 2.27: Modelo de calidad para calidad interna y externa. (NTP ISO/IEC 9126-1:2004)

2.2.4.3 Métricas del Software

Las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo del software y los proyectos de mantenimiento. (Calero, 2005; ápud Briand et al., 1996)

La métrica es "una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado". (IEEE Std 610.12-1990)

En general, la medición persigue tres objetivos: apoyar a entender qué ocurre durante el desarrollo y mantenimiento, permitirnos controlar qué ocurre en nuestros proyectos y poder mejorar nuestros procesos y productos (Fenton y Pfleeger, 1997).

Para Rodríguez y Harrison (1997), el conjunto de métricas debe indicar que aspectos de la calidad mediremos y para quién está dirigido, el conjunto de métricas a utilizar debe ser de un modelo de calidad. Las métricas deben estar validadas teóricamente o experimentalmente. La validación experimental, se realiza mediante métodos empíricos como estadísticos para evaluar la utilidad e importancia de las métricas.

A. Atributos Internos y Externos

Los niveles de algunos atributos internos influyen en los niveles de algunos atributos externos. Ejemplo, la fiabilidad puede medirse externamente observando el número de fallas en un determinado tiempo de ejecución del software, e internamente examinando las especificaciones detalladas y el código fuente para determinar el nivel de la tolerancia a fallas. Los atributos internos serían los indicadores de los atributos externos. Un atributo interno puede influenciar a una o más características, y una característica puede ser influenciada por más de un atributo figura 2.28. En este modelo la totalidad de atributos de la calidad del producto software son clasificados en una estructura de árbol jerárquica de características y sub características. El nivel más alto son las características de calidad y el nivel más bajo los atributos de calidad de software. La jerarquía no es perfecta, porque algunos atributos pueden contribuir a más de una sub característica.

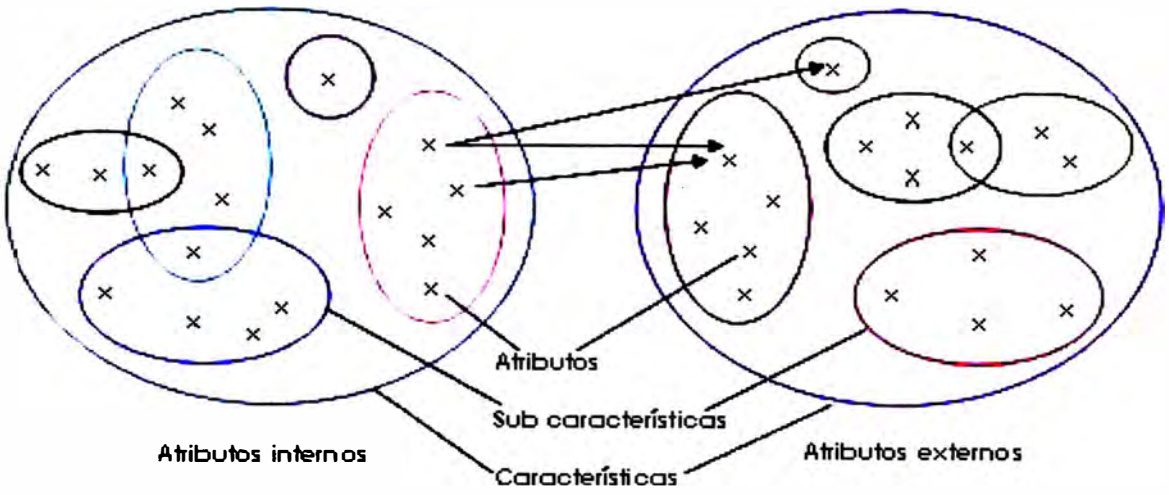

Figura N° 2.28: Características, sub características y atributos. (NTP ISO/IEC 9126-1:2004)

B. Métricas Internas

Las métricas internas se aplican a un producto software no ejecutable (una especificación o código fuente) durante el diseño y la codificación. Cuando se desarrolla un producto software, los productos intermedios deben ser evaluados usando métricas internas que permitan medir sus propiedades. El objetivo principal de las métricas internas es asegurar que se logre la calidad externa y la calidad de uso requerida. Las métricas internas proporcionan una forma de evaluar la calidad del producto software antes que el producto software sea puesto en ejecución. Las métricas internas miden atributos internos o indican los atributos externos a través del análisis de las propiedades estáticas de los entregables del software.

C. Métricas Externas

Las métricas externas usan medidas derivadas del comportamiento del software, mediante la prueba, operación y observación del software ejecutable. Las métricas externas proporcionan a los usuarios, evaluadores, testeadores y desarrolladores, el beneficio que puedan evaluar la calidad del producto software durante las pruebas u operación.

D. Relación entre las Métricas Internas y Externas

Si los requisitos de calidad del producto software esta definidos, se listan las características o sub características de calidad que contribuyen a dichos requisitos. Las métricas externas apropiadas y sus niveles son especificados

para cuantificar el criterio de calidad que valida que el software satisface las necesidades del usuario. Los atributos de calidad interna del software se definen y especifican para planear y lograr la calidad externa y calidad de uso requeridas. Métricas internas y niveles aceptables son especificados para cuantificar los atributos de calidad interna, y verificar que el software intermedio reúne las especificaciones de calidad interna.

2.2.4.4 Proceso de Evaluación del Producto Software

Tenemos una descripción de alto nivel para la calidad del producto mediante el modelo de calidad NTP-ISO/IEC 9126-1:2004, métricas externas en NTP-ISO/IEC 9126-2:2004, métricas internas en NTP-ISO/IEC 9126-3:2005, los principios de evaluación del producto software en NTP-ISO/IEC 14598-1:2005 y la evaluación del producto: proceso para desarrolladores en NTP-ISO/IEC 14598-3:2005. La aplicación de estas normas permitirá definir el proceso y la evaluación de la calidad del producto software para comercializar tara durante su desarrollo, construido mediante ICONIX descrito en la sección 2.2.2 y XP descrito en sección 2.2.3. El proceso de evaluación del producto software debe ser aplicado en forma sincronizada al desarrollo del software.

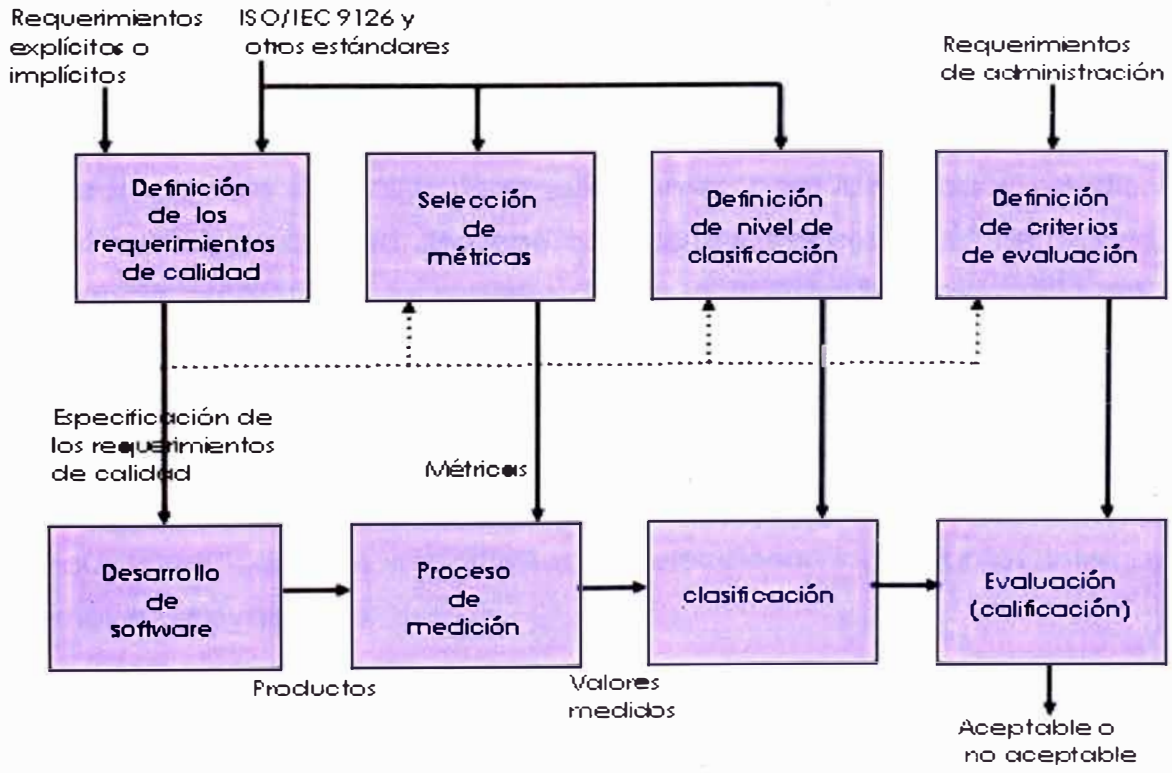

Figura N° 2.29: Proceso de evaluación del software

La tabla 2.21 muestra un modelo de medidas de calidad que relaciona las actividades del ciclo de vida del software, el modelo de referencia, los entregables factibles de ser medidos y, las métricas a usarse.

	Actividad 1	Actividad 2	Actividad 3	Actividad 4	Actividad 5
Fase	Análisis de requerimientos (Software y sistemas)	Diseño de la arquitectura (Software y sistemas)	Diseño detallado del software	Codificación y prueba del software	Integración y pruebas de calificación del software
Modelo de referencia de la serie 9126	Calidad requerida por el usuario, calidad interna requerida, calidad externa requerida	Calidad en uso pronosticado, calidad externa pronosticada, calidad interna medida.	Calidad en uso pronosticado, calidad externa pronosticada, calidad interna medida.	Calidad en uso pronosticado, calidad externa medida, calidad externa pronosticada, calidad interna medida.	Calidad en uso pronosticado, calidad externa medida, calidad externa pronosticada, calidad interna medida.
Entregables clave de la actividad	Requerimientos de calidad del usuario (especificados), requerimientos internos de calidad (especificados)	Diseño de la arquitectura de software/ sistema	Diseño detallado de software	Código de software y resultados de las pruebas	Producto software, resultado de las pruebas
Métricas usadas para medir	Métricas internas (métricas externas pueden ser usadas para validar las especificaciones)	Métricas internas	Métricas internas	Métricas internas, métricas externas	Métricas internas, métricas externas

Tabla N° 2.21: Modelo de medidas de calidad. NTP-ISO/IEC 9126-3:2005

Primer Paso: Identificación de Requerimientos de Calidad

Realizado durante el análisis de requerimientos, para identificar y estudiar los requerimientos de calidad. Se debe considerar las necesidades de usuario, la experiencia empresarial, la experiencia en la aplicación y, normas para un modelo de calidad. En consideración al párrafo anterior, determinar los pesos de las necesidades del usuario para cada característica y sub característica del modelo de calidad descrito en NTP-ISO/IEC 9126-1:2004, para la calidad interna y externa. La asignación de pesos permite centrar el desarrollo en las sub características más importantes y, seleccionar los atributos internos y externos a ser evaluados.

Segundo Paso: Especificación de la Evaluación

Aplicado durante cada actividad del proceso de desarrollo ICONIX y XP.

Identificar las métricas externas e internas a utilizar de acuerdo a NTP-ISO/IEC 9126-2:2004 y NTP-ISO/IEC 9126-3:2005, asignar niveles requeridos y registrarlos para cada sub característica de acuerdo al modelo de calidad NTP-ISO/IEC 9126-1:2004, a fin de lograr las necesidades del usuario definidas en el primer paso. Algunas especificaciones de evaluación son:

a. El desarrollador define en que actividad del ciclo de vida de ICONIX y XP, en función a la tabla 2.21 se hacen las mediciones y evaluaciones;
b. Definir las entidades a ser medidas y las métricas para cada requerimiento de calidad;
c. Definir valores objetivo para cada métrica, estos valores objetivo dan una representación cuantitativa de los requerimientos de calidad, que son usados como el criterio de evaluación;
d. Ejecutar el análisis de factibilidad de los requerimientos de calidad, si son factibles, razonables, complementarios y alcanzables.

Tercer Paso: Diseño de la Evaluación

Aplicado durante cada actividad del proceso de desarrollo ICONIX y XP. Según la NTP-ISO/IEC 9126-3:2005, desarrollar un plan de medición que contiene los entregables a ser evaluados, las métricas internas y externas a ser aplicadas, para cada sub característica del modelo de calidad elegido.

Cuarto Paso: Ejecución de la Evaluación

Aplicado durante cada actividad del proceso de desarrollo ICONIX y XP. Según la NTP-ISO/IEC 9126-3:2005, ejecutar el plan de evaluación y completar la columna resultado de la evaluación (nivel obtenido), a fin de obtener datos para la calidad del producto y comparar con los valores objetivo (nivel requerido). Los aspectos a considerar son:

a. Ejecutar el plan de medición definido en el tercer paso;
b. Obtener valores de medición reales para los atributos internos y externos de cada sub característica;
c. Promediar y almacenar la medición de los atributos internos y externos, para cada sub característica;
d. Calcular el nivel obtenido para cada característica y comparar con el nivel requerido (valor objetivo);

e. Realizar el ciclo de vida de ICONIX y XP hasta obtener el producto software, luego ponderar los valores obtenidos por sub característica y característica para obtener el resultado final e interpretar la evaluación.

2.2.5 CADENA PRODUCTIVA Y COMERCIALIZACIÓN

"Una cadena productiva es un sistema constituido por actores internos y externos interrelacionados y por una sucesión de operaciones de producción, acopio, transformación y comercialización de un producto o grupo de productos en un entorno determinado". (Van Der Heyden y Camacho, 2006)

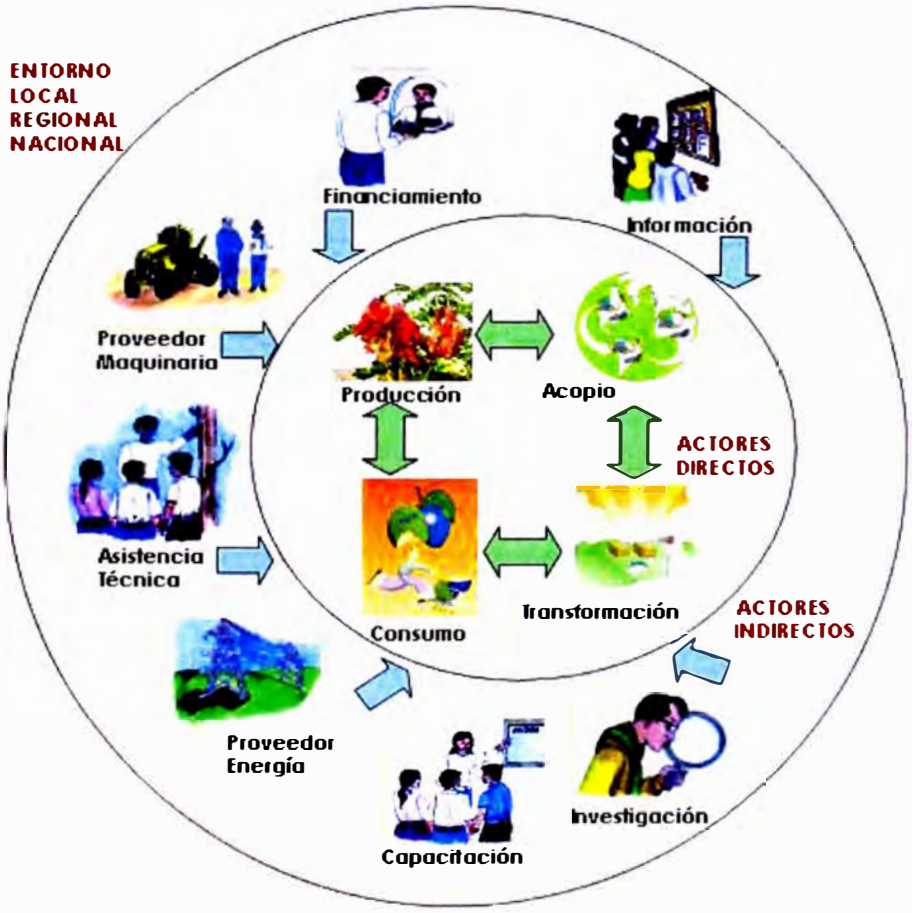

Figura N° 2.30: Esquema de una cadena productiva

Las cadenas productivas se perciben sistémicamente insertadas en la economía mundial globalizada, competitiva, innovadora y, como complejos sistemas agro alimentarios.

Los actores directos son aquellos que en algún momento, son dueños del

producto. Los actores indirectos son los que brindan bienes, insumos y/o servicios de apoyo a la cadena, pero en ningún momento tienen el producto.

Los actores directos de una cadena productiva realizan actividades interrelacionadas e interdependientes, alrededor de la evolución de un producto, desde la producción hasta su consumo. Algunos actores intervienen directamente en producir, transformar y vender un producto, otros se dedican a brindar servicios. Los actores de la cadena son afectados por el entorno, que son las condiciones ambientales o políticas.

De acuerdo a los conceptos anteriores, podemos afirmar que una cadena productiva presenta las siguientes características:

a. Sistema de actividades interrelacionadas e interdependientes.- las actividades que los actores realizan en la cadena productiva contribuyen mejorar otras. La intervención que realiza un actor (es) en un eslabón tienen relaciones de causa-efecto en la cadena, por esto, la eficiencia de la cadena esta orienta a buscar la sinergia entre los actores y cuyo efecto será la mejora del sistema en conjunto;
b. Los actores directos tienen características y roles específicos.- la presencia de actores tjene particularidades lógicas, estratégicas y de funciones, que contribuye a determinar su desempeño;
c. Competitividad para el desarrollo.- el desarrollo de la cadena supera la competitividad del sistema y se proyecta hacia el desarrollo local.

La recolección de información y análisis para una cadena productiva, esta referida al ámbito de relaciones sociales, técnicas, económicas, institucionales y culturales, considera lo siguiente: el diseño de las herramientas y métodos de recolección, la recopilación de información a nivel de los diferentes actores de la cadena, el ordenamiento, sistematización y análisis de la información por bloques temáticos (historia, entorno, actores, relaciones y organización, mercado, costos beneficios) a fin de ordenar y analizar la información de manera sistémica. (Van Der Heyden y Camacho, 2006). El anexo F, muestra un conjunto de herramientas para recolectar información sobre la comercialización de la tara y sus derivados.

A. Comercialización de Tara y su Derivados en la Región Ayacucho

Es un sistema formado por la interacción de los eslabones de producción, acopio, transformación y consumo de la tara y sus derivados, operados por los actores directos e indirectos y la necesidad de información para los actores. Existen tres objetivos estratégicos para desarrollar la cadena de tara: (1) Mejorar la oferta productiva de tara en Ayacucho, (2) Mejorar la eficiencia de la gestión empresarial de los actores, (3) Fortalecer la institucionalidad de los espacios de diálogo y concertación.

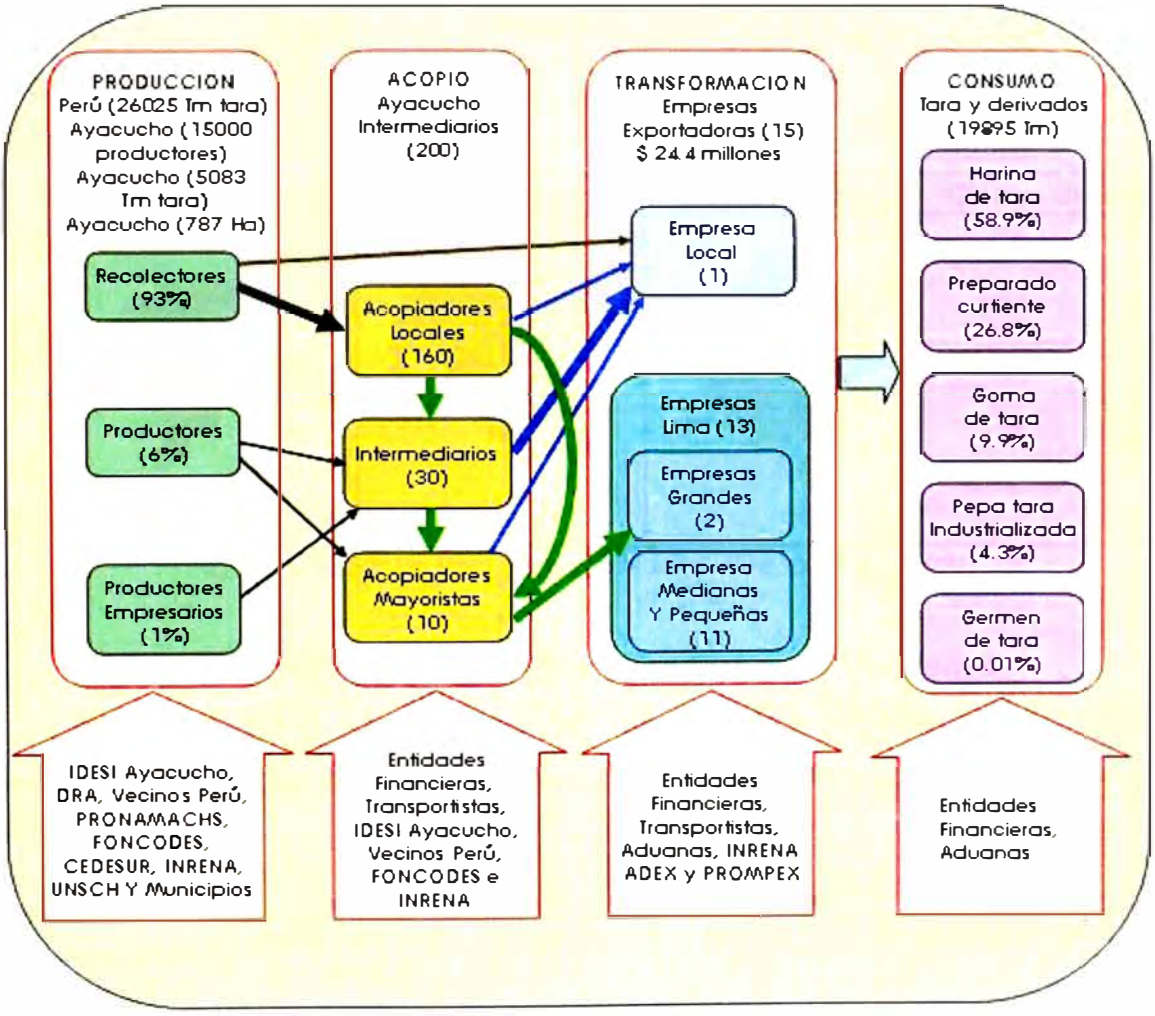

Figura N° 2.31: Flujo de la cadena productiva de tara. (Avendaño, et al., 2007)

B. La Tara y sus Derivados

La vaina de tara proporciona derivados con potencial industrial, alimenticio y médico, para producir hidrocoloides (gomas), taninos y ácido gálico, entre otros. La vaina esta compuesta por semilla y cáscara, esta presenta mayor concentración de taninos, usado en la industria de: cueros,

plásticos, adhesivos, galvanizados, vinos, pinturas, cerveza. El ácido gálico derivado del tanino es usado como: antioxidante para aceites, decolorante en la industria de cervecera, en fotografía, tintes, fábrica del papel, litografía. Las semillas tienen como composición: cáscara (28%), gomas (34%) y germen (37.5%), esta parte del fruto es usado para: producir aceite, helados, harina proteica. Existen otros derivados de la semilla para producir: jabones, pinturas, barnices, esmaltes, tintes de imprenta, mantecas y margarinas comestibles, por su bajo contenido de ácido oleico (1.4%). El uso medico en: gastroenterología, úlceras, cicatrizantes, antinflamatorios, antisépticos, antidiarreicos, antimicóticos, antibacterianos, antiescorbúticos, odontológicos. (De la Cruz, 2004)

Vaina de Tara	Semilla de Tara	Hojuela Tara	Goma de Tara
			

Figura N° 2.32: La tara y sus derivados. (De la Cruz, 2004)

C. Características Físicas y Químicas de la Tara y sus Derivados

Presentamos la tabla 2.21 que muestra las características físicas y químicas, indicando el procedimiento para obtener la composición en porcentaje.

Característica	Fruto (vaina y semilla) (%)	Semilla (%)	Goma (%)	Germen (%)	Cáscara (%)
Humedad	11.70	12.01	13.76	11.91	10.44
Proteínas	7.17	19.62	2.50	40.22	1.98
Cenizas	6.24	3.00	0.53	8.25	3.05
Fibra bruta	5.30	4.00	0.86	1.05	1.05
Estracto etéreo	2.01	5.20	0.48	12.91	0.97
Carbohidratos	67.58	56.17	81.87	25.66	83.56
Taninos	62.00	-.-	-.-	-.-	-.-
Azucares	-.-	-.-	83.20	-.-	-.-

Tabla N° 2.22: Características de la tara y sus derivados. (De la Cruz, 2004)

Humedad.- contenido de humedad del fruto o derivados bajo condiciones de temperatura y presión.

Proteína.- determinado empleando el método Kjeldhal, utilizando como catalizador selenio, con factor de conversión de proteínas 6,25.

Extracto etéreo.- determinando usando el método Boxhlet, con tiempo de extracción 6 horas.

Cenizas.- determinando por el método de incineración a la temperatura de 550 °C, durante 6 horas.

Fibras brutas.- residuo orgánico lavado y seco que queda después de hervir sucesivamente el material con H_2SO_4 y NaOH y, convertirlo en ceniza.

Carbohidratos.- determinado por diferencia de los análisis de humedad, proteína, cenizas, fibra bruta y extracto etéreo.

Azúcares totales.- determinado por el método volumétrico de Lane Eynon, consiste en agregar la solución hidrolizada de goma a un volumen determinado de solución de Fehling, a fin de reducir todo el ión cúprico o cuproso.

Fibra dietética.- La muestra es gelatinizada y digestada enzimáticamente con proteasa y amiglucosidasa para remover la proteína y el almidón. Se agrega cuatro volúmenes de 60 ml de etanol al 95% para precipitar la fibra soluble. El precipitado es filtrado, secado y pesado.

2.2.6 ARQUITECTURA TECNICA (AT)

Según Rosenberg y Stephens (2007), la AT abarca tres áreas: (1) modelo de despliegue (servidores, comunicaciones, topología del sistema, navegadores Web, etc.), (2) modelo de paquetes/componentes, (3) modelo de datos. Diseñar una arquitectura técnica (arquitectura del sistema y arquitectura del software) implica tener una visión general del software a desarrollar, a fin seleccionar la tecnología para la implementación en relación a los requisitos. La AT describe el sistema en términos de estructura, incluye la topología del sistema (servidores, ubicación física de la red, etc.). Para diseñar una buena AT considerar: accesos simultáneos al software, horas pico de uso, número de transacciones por minuto, posibles fallas, escalabilidad, seguridad, disponibilidad, localización, auditoria, portabilidad e implementación. Existen razones para seleccionar el framework Spring MVC, que son las siguientes:

a. El framework Spring MVC es open source que lo hace portable a otros IDE como NetBeans, Eclipse o SpringCore comparado con ASP.net.
b. Spring MVC presenta una división clara entre el modelo (JavaBeans), vistas (Java Server Pages - JSP) y controladores (Servlets).
c. Spring MVC es flexible, porque implementa toda su estructura mediante interfaces, no como Struts que obliga a heredar de clases concretas sus Actions y Forms.
d. Spring MVC permite utilizar JSP, XLST, Velocity, FreeMaker u otro lenguaje para integrarlo a la vista de una aplicación, lo que no permite el JSF (Java Server Pages).
e. Spring MVC tiene controladores que se configuran mediante inversión de control (IoC) e inyección de dependencia (ID) con los demás objetos, haciendo fácil probar e integrar con otros objetos del contexto de Spring.
f. Spring MVC presenta componentes fácilmente testeables comparado con Struts, porque evita la herencia de una clase de manera forzada y tiene dependencia directa en el control del servlet que despacha las peticiones.
g. La capa Web de Spring es una pequeña parte en la capa de negocio de Spring, siendo una buena práctica. Struts y otros frameworks Web dejan a elección la implementación de los objetos de negocio.
h. Spring MVC no posee ActionForms, que es propio de Struts, enlazando directamente los controladores con los beans de negocio.
i. Spring MVC presenta más código testeable que otros frameworks como el Struts o JSF. Las validaciones no dependen de la API de Servlets, son independientes.
j. Struts obliga a extender la clase Action, mientras que Spring MVC no, proporcionando una serie de implementaciones en el controlador para que el usuario los pueda utilizar.
k. Spring MVC tiene una interfaz bien definida para la capa de negocio lo que no posee Struts o JSF.

FRAMEWORK SPRING

Al diseñar la AT para sistemas basados en Web, definir el framework Web a utilizar y el lenguaje de programación. De acuerdo al plan de investigación usaremos java y el framework spring[1], específicamente parte del framework spring denominado Spring Web MVC. Esta parte de spring permite crear una

1 Vea http://www.springframework.org.

aplicación Web usando la arquitectura Modelo-Vista-Controlador (MVC), para usar las características Web MVC, ejecutarlo en un contenedor java servlet/SP como Tomcat.[2]

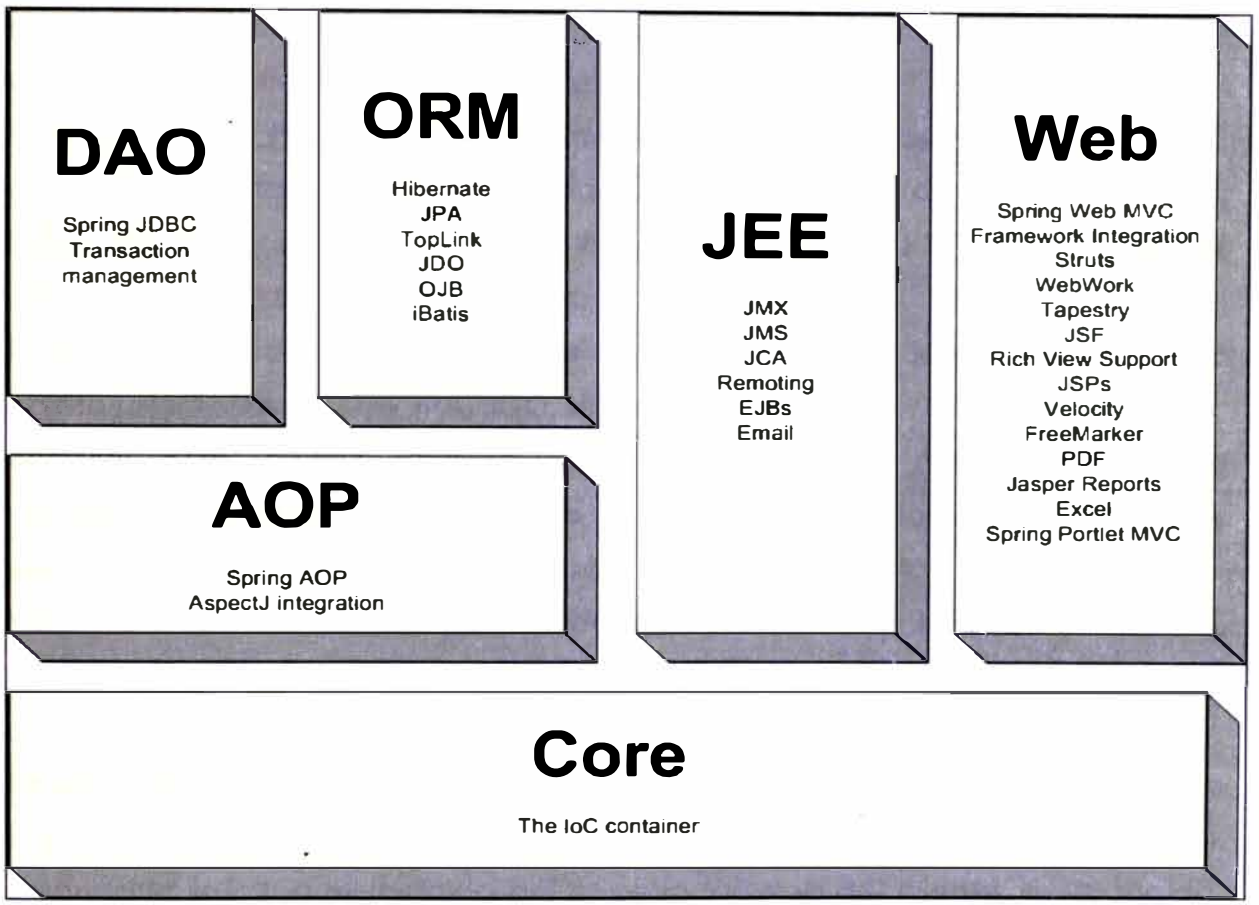

Figura N° 2.33: Arquitectura del framework spring. (Johnson et al., 2008)

A. Arquitectura del Framework Spring

El framework spring contiene siete módulos como se observa en la figura 2.33 y es ligero para aplicaciones J2EE, no se limita necesariamente a J2EE, spring se usa en el servidor.

Paquete Core.- Es la parte fundamental de spring, proporciona el patrón de diseño inversión de control (IoC) y las características de inyección de dependencia (Johnson et al., 2008). La IoC permite escribir una clase como un JavaBean "puro" con sus propias propiedades, pero sin dependencia del framework en el que está corriendo. Al usar IoC un objeto ("bean") no busca

2 Vea http://jakarta.apache.org/tomcat.

datos, los objetos dependientes son entregados al "bean" mediante el framework.

Paquete Context.- Esta construido sobre la base del paquete Core. El paquete de contexto hereda sus características del paquete de beans y añade soporte para la internacionalización, propagación de eventos, carga de recursos y, creación de contextos como el servlet container.

Paquete DAO.- Paquete que provee una capa de abstracción de JDBC, elimina la necesidad de realizar código y análisis (parsing) de la base de datos. El soporte spring JDBC, proporciona el control "final" sobre la forma en que los objetos persisten en la base de datos.

Paquete ORM.- Provee capas para la integración con los API's de mapeo objeto-relacional, incluye JPA, JDO, Hibernate e Ibatis.

Paquete AOP.- Spring proporciona el paquete AOP compatible con la programación orientada a aspectos, por ejemplo métodos de intercepción y poincuts. La implementación se realiza a nivel de metadatos.

Paquete Web.- Provee integración de las características básicas de la Web, inclusive funcionalidades de upload de archivos y la inicialización del IoC container usando servicios como servlets.

Paquete MVC.- Este paquete es parte de Spring Web MVC que proporciona la implementación del Modelo-Vista-Controlador (MVC) para aplicaciones Web. Cuando la solicitud es recibida del navegador Web del cliente, es "recogida" por el servidor Web (Tomcat). Tomcat pasa la solicitud al DispatcherServlet, que luego maneja la solicitud en una de las clases controlador del software. En MVC la aplicación se divide en tres áreas distintas:

Modelo.- Este es un objeto que representa los datos, generalmente al leer una base de datos. La sesión "detrás" del modelo esta detallado para el mapeo de objetos a tablas, filas, columnas y, relaciones en la base de datos.

Vista.- Es el límite entre la computadora y el usuario, en una aplicación Web, la vista es la página Web y la plantilla (JSP) que crea la página Web. Una fortaleza de spring es la separación de la vista del resto del framework MVC, porque podemos elegir la tecnología de vista como: Tiles, Velocity, XSLT, Acrobat, Excel, etc.

Controlador.- Los controladores son el "pegamento" entre la vista y el modelo. Cuando una solicitud es recibida, el controlador busca (o actualiza) la

información del modelo, decide que vista mostrar al usuario y, maneja los datos necesarios para la vista. Normalmente, una aplicación MVC tiene muchos controladores, los controladores pueden contener la lógica de la aplicación y la lógica del negocio, que pueden estar separados en diferentes capas. Los controladores en spring devuelven un objeto ModelAndView, que le dice a spring que vista devolver al navegador y con qué modelo de datos poblarlo.

B. Arquitectura de Aplicaciones Construidas en Spring

Una aplicación Web en spring implementa los componentes de acuerdo a la figura 2.34, como sigue:

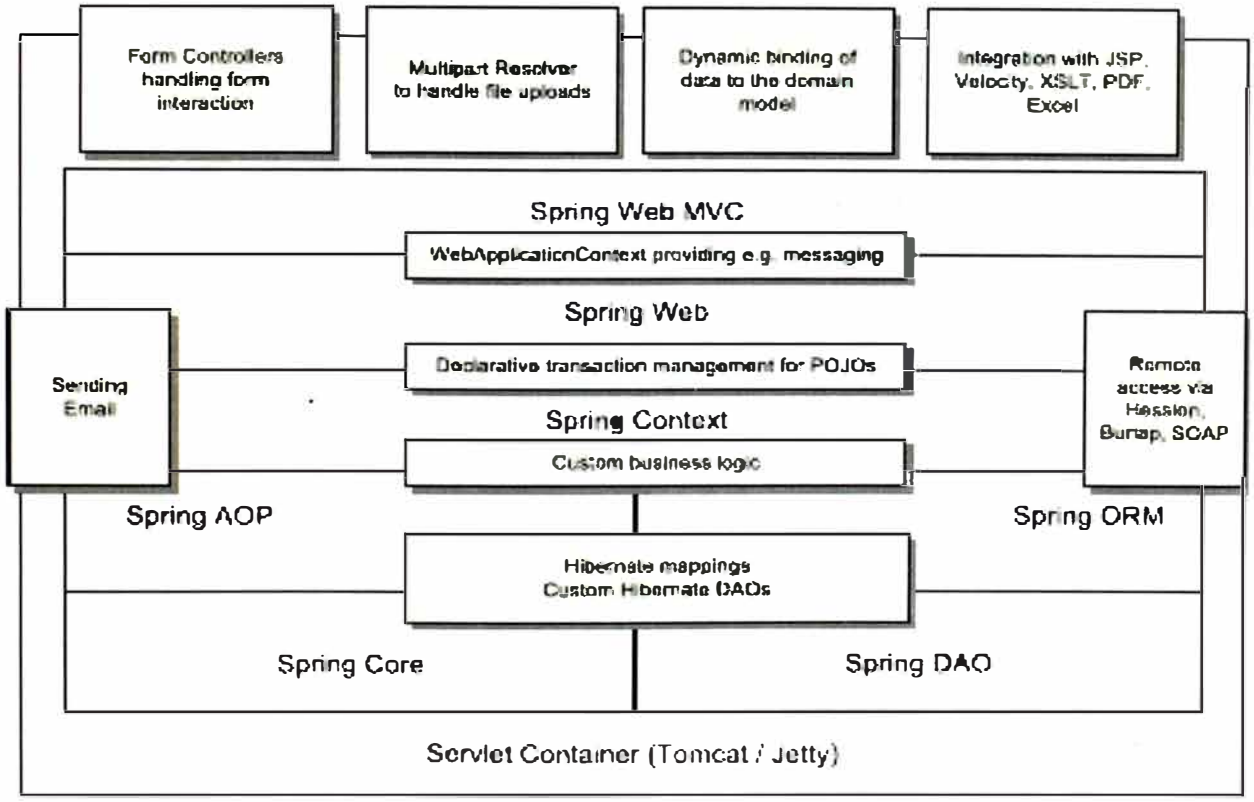

Figura N° 2.34: Arquitectura típica de aplicación Web en spring. (Johnson et al., 2008)

C. Inyección de Dependencia

Es una especialización de IoC, la inyección de dependencia es una técnica que los framework utilizan como conductor en la aplicación, el framework realiza la labor de conectar las dependencias de aplicación, removiendo de la aplicación todo el código hacia la lógica y manejo de la

creación de objetos de la aplicación. (Ladd et al., 2006)

D. Peticiones Web (Request Processing)

Spring al igual que otros frameworks maneja las peticiones alrededor de un servlet central que despacha las solicitudes a los controladores y facilita el flujo web de la aplicación.

DispatcherServlet.- Es el servlet central del framework, además esta integrada con el IoC Container y como tal permite usar cualquier otra característica propia del contenedor.

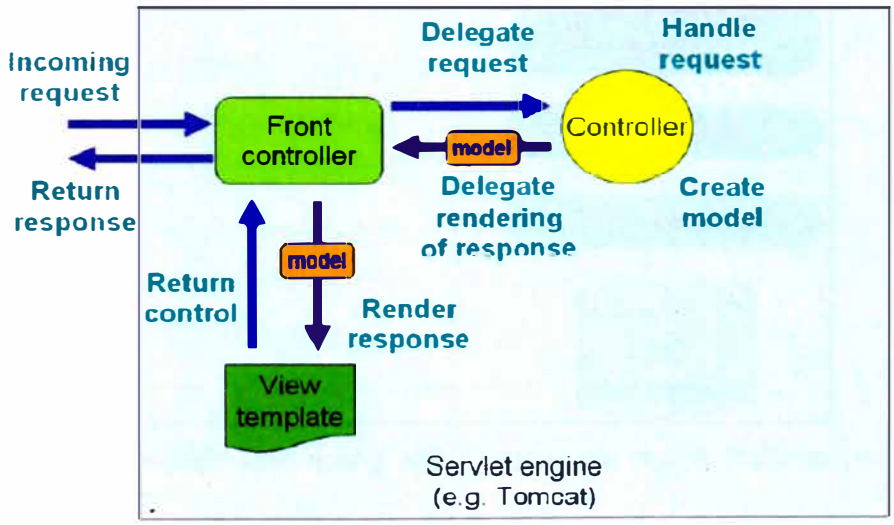

Figura N° 2.35: Flujo de la petición Web en Spring Web MVC. (Johnson et al., 2008)

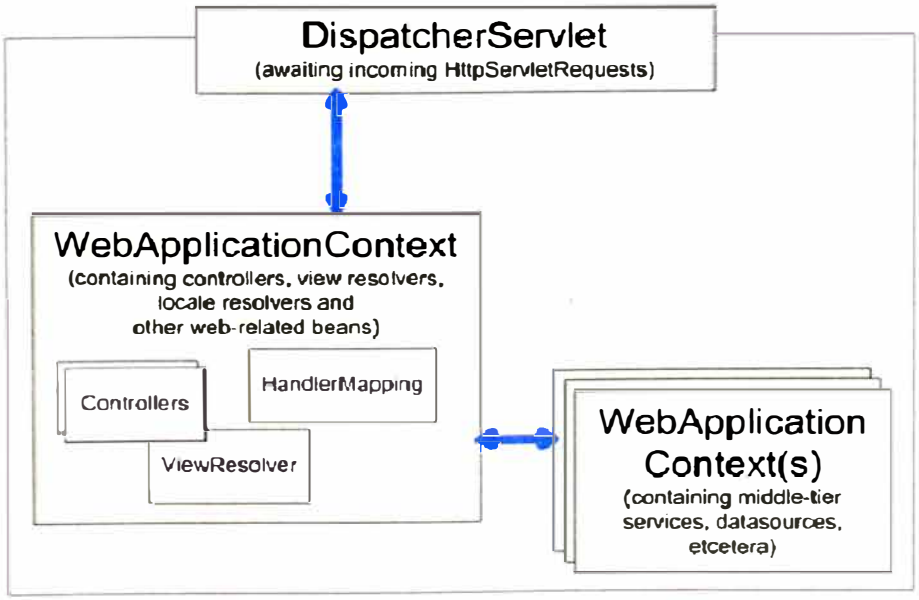

Figura N° 2.36: Jerarquía de contextos en Spring MVC. (Johnson et al., 2008)

E. Spring JDBC y PostgreSql

Este es el esquema de implementación del acceso a base de datos en spring, la base de datos es PostgreSql (versión 8.3.6.1). Una de las razones porque elegimos spring, es que permite crear un modelo de objetos persistentes usando JavaBeans, con métodos simples como get y set para cada propiedad. Estos JavaBeans están en nuestras clases de dominio y, existe un mapeo directo entre éstos y las clases del modelo de dominio.

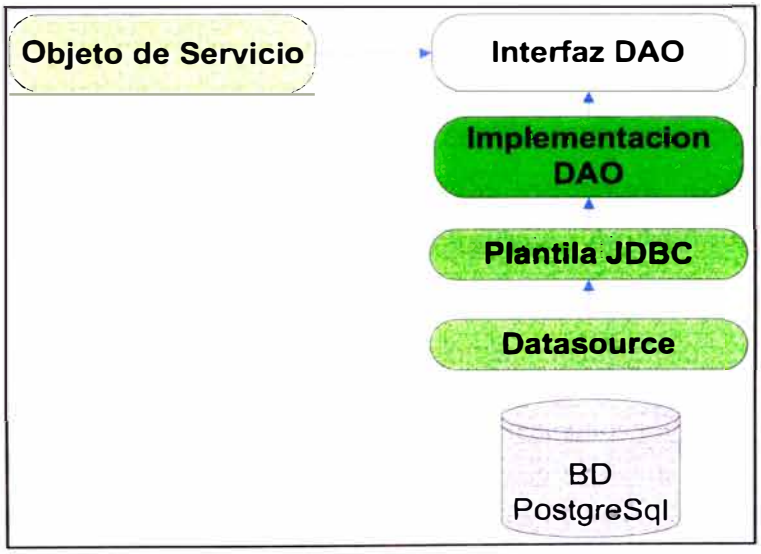

Figura N° 2.37: Conexión con spring JDBC a base de datos. (Johnson et al., 2008)

CAPITULO III

METODOLOGIA DE LA INVESTIGACION

3.1 TIPO DE INVESTIGACION

Desarrollaremos un producto software, aplicando las metodologías ICONIX y XP, luego medir, comparar y evaluar la calidad del producto software usando un modelo de calidad que incorpore características, sub características y atributos, utilizando métricas de calidad interna y externa. Desde el punto de vista científico, describir es medir (Hernández et al., 1997), por esta consideración el tipo de investigación es descriptiva.

3.2 DISEÑO DE LA INVESTIGACION

Según Hernández et. al (1997) las investigaciones no experimentales son "aquellas que se realizan sin manipular deliberadamente las variables, es decir, no se varía intencionalmente la variable independiente, simplemente lo que se hace es observar las funciones tal y como se dan en su contexto natural para después analizarlo". En la presente investigación usamos los procesos de ingeniería para construir software ICONIX y XP, para luego medir la calidad del producto software mediante un estándar, sin variar las variables de investigación.

De acuerdo a Hernández et. al (1997) una investigación de diseño transversal es "cuando se recolectan datos en un solo momento en un tiempo único y su propósito es describir variables y analizar los hechos tal y como se dan". El instrumento de recolección de datos se usa durante el desarrollo del software para cada proceso (ICONIX y XP) y de forma única. Por las consideraciones anteriores el diseño de la investigación es no experimental y transversal.

3.3 POBLACION Y MUESTRA

Población.- Los procesos para desarrollo de software usando las metodologías

ágil y formal ICONIX y ágil XP.

Muestra.- Un proceso de desarrollo de software usando la metodología ágil y formal ICONIX y otro proceso de desarrollo de software aplicando la metodología ágil XP.

3.4 VARIABLES E INDICADORES

Variable Independiente

X: Metodología para desarrollo de software.- Son procesos de ingeniería compuesto por métodos, técnicas y herramientas necesarias para construir software de alta calidad.

Indicadores de la Variable Independiente

X1: ICONIX.- Es una metodología simplificada que unifica un conjunto de métodos orientados a objetos, consiste en producir artefactos desarrollados incrementalmente y en paralelo donde el modelo estático se incrementa y refina al modelo dinámico. Presenta características de ser pequeño y simple, conducido por casos de uso, usa algunos modelos del UML y, presenta alto grado de trazabilidad. Compuesto por las fases de análisis de requisitos, diseño preliminar, diseño detallado, implementación y, pruebas.

X2: Programación Extrema (XP).- Es una metodología ligera, eficiente, flexible y de bajo riesgo para equipos pequeños y medios. Basado en los valores: simplicidad, comunicación, retroalimentación y coraje. Para desarrollar el software utiliza doce prácticas. Presenta las fases de: exploración, planificación, iteración, producción, mantenimiento y retiro. Se diferencia de otras metodologías por su realimentación continua y ciclos cortos, aplicar planificación incremental, flexibilidad para implementar la funcionalidad, pruebas automatizados escritos por programadores y usuarios, comunicación oral permanente, colaboración estrecha entre programadores, prácticas de corto plazo de los programadores e intereses de largo plazo del proyecto.

Variable Dependiente

Y: Valoración del uso de la metodología.- Mediante las características y sub características del modelo de calidad interna y externa para el producto

software de acuerdo a la norma NTP ISO/IEC 9126-1:2004.

Indicadores de la Variable Dependiente

Y1: Funcionalidad.- La capacidad del producto software para proveer las funciones que satisfacen las necesidades explícitas e implícitas cuando el software se utiliza bajo condiciones específicas.

Y2: Flabilidad.- La capacidad del producto software para mantener un nivel específico de funcionamiento cuando se está utilizando bajo condiciones especificadas.

Y3: Usabilidad.- La capacidad del producto software de ser entendido, aprendido, usado y atractivo al usuario, cuando es usado bajo las condiciones especificadas.

Y4: Eficiencia.- La capacidad del producto software para proveer un desempeño apropiado, de acuerdo a la cantidad de recursos utilizados y bajo las condiciones planteadas.

Y5: Facilidad de mantenimiento.- Capacidad del producto software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, y en requerimientos y especificaciones funcionales.

Y6: Portabilidad.- La capacidad del software para ser trasladado de un entorno a otro.

3.5 TECNICAS E INSTRUMENTOS

3.5.1 INSTRUMENTO PARA RECOLECTAR INFORMACION DE COMERCIALIZACION DE TARA

Se uso las técnicas de entrevista, encuesta a los actores directos e indirectos de la cadena y, revisión documentaría para recolectar datos e información, se presenta los instrumentos de forma sistematizada en el anexo E, información necesaria para comprender en profundidad la comercialización de tara en la Región Ayacucho.

Caracterización de producción.- Modela el levantamiento de información sobre factores de producción delimitados a cada provincia, específicamente por zonas (altitud, clima, etc.) cuyo objetivo es informar sobre la oferta productiva potencial de la tara en la Región.

Inventario de actores directos e indirectos.- Modela la información necesaria sobre los actores clasificados por eslabones, tipo de actor y, su ubicación por localidad, distrito y provincia en la Región. Asimismo, se caracteriza a los actores directos sobre sus actividades, organización y resultados económicos a los actores indirectos sobre sus actividades, la oferta de bienes y servicios, la calidad sus bienes y servicios, sus costos, etc.

Relaciones entre actores directos.- Formato para recolectar información sobre la interacción de acopiadores locales y los actores directos de los otros eslabones, este formato debe adaptarse para los actores de los otros eslabones. La información recolectada presenta las transacciones que realiza el acopiador local, poder de negociación y, modalidad del pago.

Matriz de competencia.- Herramienta formulada para levantar información sobre la producción de tara por provincias, aplicada a los productores y expertos, que resume la capacidad de producción, calidad, precios, costos, meses de cosecha y mercados.

Demanda y mercados.- Formato para recolectar información entrevistando a un grupo de actores que realizan actividades de comercialización, orientada al mercado (local, regional, nacional), meses de mercado, forma de pago, precio promedio, proveedor (productor, acopiador, transformador) y volumen.

Matriz histórica de precios.- Formato para obtener la variación del precio promedio por años y provincias, información obtenida entrevistando a un grupo de actores que comercializan tara.

3.5.2 INSTRUMENTO PARA EVALUAR LA CALIDAD DEL PRODUCTO SOFTWARE

Para recolectar información relacionada a la evaluación de la calidad del producto software, usamos las técnicas de entrevistas, encuestas y análisis de información que permita definir los pesos de las características y sub características, métricas ha ser aplicadas y el nivel requerido para cada sub característica. Estos valores serán los mismos para las metodologías ICONIX y XP, porque son requerimientos de calidad de los usuarios, a fin de tener una

base de comparación común, como se presenta las tablas 3.1 a 3.4.

CARACTERISTICA	PESO CARACTERISTICA
Funcionalidad	
Fiabilidad	
Usabilidad	
Eficiencia	
Facilidad de Mantenimiento	
Portabilidad	

Tabla Nº 3.1: Pesos de características de las necesidades de usuarios

CALIDAD EXTERNA E INTERNA		
CARACTERISTICA	**SUB CARACTERISTICA**	**PESO SUB CARACTERISTICA**
Funcionalidad	Aplicabilidad	
	Precisión	
	Interoperatividad	
	Seguridad	
Fiabilidad	Madurez	
	Tolerancia a fallas	
	Recuperabilidad	
Usabilidad	Entendibilidad	
	Facilidad de aprendizaje	
	Operabilidad	
	Atractividad	
Eficiencia	Comportamiento en el tiempo	
	Utilización de recursos	
Facilidad de Mantenimiento	Analizabilidad	
	Cambiabilidad	
	Estabilidad	
	Testeabilidad	
Portabilidad	Adaptabilidad	
	Instalabilidad	
	Coexistencia	
	Reemplazabilidad	

Tabla Nº 3.2: Pesos de sub características de las necesidades de usuarios

CARACTERISTICA	SUB CARACTERISTICA	METRICAS	NIVEL REQUERIDO
Funcionalidad	Aplicabilidad		
	Precisión		
	Interoperatividad		
	Seguridad		
Fiabilidad	Madurez		
	Tolerancia a fallas		
	Recuperabilidad		
Usabilidad	Entendibilidad		
	Facilidad de aprendizaje		
	Operabilidad		

	Atractividad		
Eficiencia	Comportamiento en el tiempo		
	Utilización de recursos		
Facilidad de Mantenimiento	Analizabilidad		
	Cambiabilidad		
	Estabilidad		
	Testeabilidad		
Portabilidad	Adaptabilidad		
	Instalabilidad		
	Coexistencia		
	Reemplazabilidad		

Tabla Nº 3.3: Categoría de medición de calidad interna y externa

SUB CARACTERISTICA	ENTREGABLES A SER EVALUADOS	METRICAS INTERNAS A SER APLICADAS	METRICAS EXTERNAS A SER APLICADAS
1. Aplicabilidad	1. 2.	1. 2.	1. 2.
2. Precisión			
3. Interoperatividad			
4. Seguridad			
5. Madurez			
6. Tolerancia a fallas			
7. Recuperabilidad			
8. Entendibilidad			
9. Facilidad de aprendizaje			
10. Operabilidad			
11. Atractividad			
12. Comportamiento en el tiempo			
13. Utilización de recursos			
14. Analizabilidad			
15. Cambiabilidad			
16. Estabilidad			
17. Testeabilidad			
18. Adaptabilidad			
19. Instalabilidad			
20. Coexistencia			
21. Reemplazabilidad			

Tabla Nº 3.4: Plan de medición

3.5.3 HERRAMIENTAS PARA EL TRATAMIENTO DE DATOS E INFORMACION

Las herramientas tecnológicas ha utilizar, se seleccionan en función a las limitaciones existentes para el proyecto: ser una aplicación orientada a una PYME, software pequeño o mediano, existe escasos recursos humanos y financieros para el desarrollo, existe limitación para implementar la infraestructura física y tecnológica del ambiente de producción del software. Considerando estos aspectos definimos las tecnologías como observamos en

la tabla 3.5.

Nombre	Fabricante	Licencia	Servicio
Windows XP SP2	Microsoft Corporation	Licencia propietaria	Sistema Operativo
Sun GlassFish Enterprise Server Ver. 2.1	Sun Microsystems	CDDL* GNU GPL**	Servidor de aplicaciones, implementa y ejecuta aplicaciones desarrolladas con J2EE
Java Plataform Enterprise Edition (JEE) Ver. 6.14	Sun Microsystems	CDDL* GPLv2	Plataforma de programación, para desarrollar y ejecutar software en lenguaje de programación Java, usa arquitectura de N niveles distribuida, ejecutándose sobre un servidor de aplicaciones
HTML (Lenguaje de Marcas de Hipertexto) Ver. 4.01	World Wide Web Consortium (W3C)		Lenguaje para construir páginas con texto e imágenes Web. HTML puede incluir un script (Javascript) que afecta el comportamiento de navegadores Web y otros procesadores de HTML
JavaScript Ver. 1.8	Netscape Comunica-tions, Mozilla Foundation		Lenguaje de programación orientado a objetos interpretado, usado en páginas Web
Spring framework Ver. 2.5	SpringSource	Apache License 2.0	Framework de código abierto para desarrollo de aplicaciones Java del lado servidor y otras, con Spring es posible utilizar JavaBeans sencillos
Junit framework Ver. 4.3.1	JUnit Yahoo Group	IBM Common Public License	Framework para evaluar el funcionamiento de una clase y sus métodos Java
PostgreSQL Ver. 8.4	PostgreSQL Global Development Group	BSD (Berkeley Software Distribution)	Sistema gestor de bases de datos objeto-relacional de la Universidad de Berkeley, incluye algunas características de orientación a objetos como herencia
NetBeans IDE 6.7 (Build 200906241340)	Sun Microsystems	CCDL*	IDE de código abierto multiplataforma para desarrollar "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.
Enteprise Architect Ver. 7.1	Sparx Systems	Licencia propietaria	Entorno de modelado para la metodología ICONIX
Microsoft Office FrontPage 2003	Microsoft Corporation	Licencia propietaria	Entorno para diseño de interfaces Web

Tabla Nº 3.5: Herramientas tecnológicas para el tratamiento de datos.

* CDDL: Licencia Común de Desarrollo y Distribución
** GPL: Licencia Pública General de GNU

3.5.4 TECNICA PARA APLICAR ICONIX

Luego de revisar el marco teórico desarrollado en el capitulo II, sección 2.2.2, formulamos el proceso considerando las fases e hitos para desarrollar el software, usando la metodología ICONIX, de acuerdo a las tablas 3.6 a 3.13.

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Identificar requisitos	• Requisitos funcionales y no funcionales • Casos de prueba	• Entrevistas • Definir lo que el sistema debe hacer • Escribir al menos un caso de prueba para cada requisito	Usuario Cliente Analista
Identificar objetos del mundo real y dibujar modelo de dominio	Modelo de dominio	• Identificar clases clave del negocio • Identificar sustantivos y depurar • Identificar objetos en requisitos funcionales y asignar al modelo de dominio • Utilizar agregación y generalización	Analista
Realizar prototipo de interfaz gráfica	Prototipo GUI	• Utilizar historia de eventos del usuario (storyboards) • Utilizar los requisitos funcionales • Diseñar interfaz grafica básica	Programador Analista
Descubrir casos de uso	Lista de casos de uso	• Utilizar requisitos funcionales • Entrevistas	Usuario Cliente Analista
Dibujar y empaquetar casos de uso	• Diagrama de casos de uso • Paquete de casos de uso	• Identificar roles y responsabilidades de actores • Asociar actores con casos de uso • Relacionar casos de uso • Agrupar lógicamente casos de uso	Analista
Asignar requisitos funcionales a los casos de uso	Relación entre requisitos funcionales y casos de uso	• Asignar requisitos funcionales a los casos de uso	
Escribir el primer borrador de casos de uso	Primer borrador de casos de uso	• Utilizar glosario de objetos del modelo de dominio • Utilizar la regla de dos párrafos • Escribir el caso de uso como flujos de evento/respuesta • Escribir el caso de uso con estructura sustantivo-verbo-sustantivo • Escribir caso de uso en voz activa • Referenciar por su nombre las pantallas	

Tabla Nº 3.6: Análisis de requisitos

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Revisar el modelo de dominio	Modelo de dominio	• Identificar al menos 80% de clases clave de dominio del problema	Usuario Cliente Analista Programador
Revisar el prototipo GUI	Prototipo GUI	• Diseñar con precisión la GUI relacionada al caso de uso	
Revisar modelo de casos de uso	Primer borrador de casos de uso	• Eliminar clases fuera del dominio del problema • Cambiar descripción de voz pasiva activa • Describir todos los cursos alternos • Asociar todos los requisitos a las casos de uso • Describir que intenta hacer el usuario para cada caso de uso	

Tabla N° 3.7: Revisión de requisitos (primer hito)

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Rescribir el primer borrador para cada caso de uso	Caso de uso desambiguado	• Rescribir el caso de uso durante el análisis de robustez	Analista
Identificar el primer corte de objetos que completan escenarios para cada caso de uso	Diagrama de robustez	• Copiar la descripción del caso de uso en el diagrama de robustez. • Usar las clases del modelo de dominio • Crear un objeto interfaz por cada GUI y nombrarlo • Transformar verbos del caso de uso en controladores • Relacionar un caso de uso al diagrama de robustez cuando es invocado • Utilizar las reglas para construir el diagrama de robustez	
Actualizar el modelo de dominio	Modelo de dominio actualizado	• Actualizar el modelo de dominio con nuevas clases y atributos durante el análisis de robustez.	
Actualiza el diagrama de clases de análisis	Modelo de dominio actualizado	• Actualizar el diagrama de clases de análisis al finalizar el análisis de robustez • Asignar atributos a las clases entidad	

Tabla N° 3.8: Diseño preliminar

Revisar descripción de casos de uso	Caso de uso	• Coincidir la descripción del caso de uso con el diagrama de robustez	Usuario Cliente Analista Programador
Revisar diagrama de robustez	Diagrama de robustez	• Coincidir el diagrama de robustez con descripción del caso de uso • Verificar que el diagrama de robustez cumple las reglas • Verificar que el diagrama de robustez tiene todos los cursos alternos	
Revisar modelo de dominio actualizado	Modelo de dominio actualizado	• Coincidir los objetos entidad del diagrama de robustez con el modelo de dominio actualizado	

Tabla Nº 3.9: Revisión de diseño preliminar (segundo hito)

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Diseñar diagrama de componentes	Diagrama de componentes	• Entrevistas • Características del negocio • Utilizar la arquitectura MVC • Integrar frameworks	Cliente Analista Programador Arquitecto de software
Diseñar diagrama de despliegue	Diagrama de despliegue	• Entrevistas • Características del negocio	

Tabla Nº 3.10: Arquitectura técnica

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Dividir modelo de dominio actualizado para cada caso de uso	Parte de modelo de dominio actualizado	• Coincidir las clases entidad del diagrama de robustez con parte del modelo de dominio actualizado y dibujarlo	Diseñador
Dibujar un diagrama de secuencia para cada caso de uso	Diagrama de secuencia	• Copiar la descripción del caso de uso • Copiar objetos entidad, interfaz y actores del diagrama de robustez • Verificar que un mensaje del diagrama de secuencia es verbo en el caso de uso • Hacer refactoring al diagrama de secuencia antes de codificar	Programador Diseñador
Actualizar el diagrama de clases de un caso de uso	Diagrama de clase	• Asignar operaciones a las clases a partir de mensajes del diagrama secuencia • Establecer multiplicidad en las clases • Depurar las clases, operaciones y atributos del diagramo de clases	

		diagrama de clases	
Extraer controladores para pruebas unitarias	Lista de controladores	• Identificar controladores para la lógica del negocio desde un diagrama de robustez	

Tabla N° 3.11: Diseño

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Revisar diagrama de secuencia	Diagrama de secuencia	• Verificar que el diagrama de secuencia coincide con la descripción del caso de uso • Verificar que el diagrama de secuencia representa los cursos básico y alterno • Verificar en los mensajes que los atributos y valores de retorno son correctos	Diseñador Programador
Revisar diagrama de clases	Diagrama de clases	• Verificar que el nombre, atributos y operaciones se asignaron correctamente a las clases • Asignar requisitos no funcionales a los casos de uso y clases	
Revisar modelo dominio actualizado	Modelo de dominio actualizado	• Verificar nombres y atributos del modelo dominio actualizado	
Revisar lista de pruebas unitarias	Lista de controladores	• Actualizar la lista de controladores	

Tabla N° 3.12: Revisión critica de diseño (tercer hito)

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Implementar la base de datos física	Base de datos física	• Escribir el script usando el modelo de dominio actualizado • Ejecutar el script usando un DBMS	Programador
Implementar código para clases entidad	Código fuente	• Escribir o generar código fuente con una herramienta usando el modelo de dominio actualizado	Programador
Implementar código para las interfaces	Código fuente	• Generar código fuente usando una herramienta	Programador
Crear pruebas unitarias para cada controlador	Prueba unitaria	• Escribir código fuente para una prueba unitaria usando una herramienta	Programador
Implementar código fuente para cada controlador	Código fuente	• Escribir el código fuente siguiendo el flujo normal del diagrama de secuencia usando una herramienta	Programador

		• Actualizar el diagrama de secuencia con la codificación	
Ejecutar pruebas unitarias para cada controlador	Reporte de pruebas unitarias	• Ejecutar el modulo de cada prueba unitaria • Modificar código fuente si la prueba unitaria muestra resultado incorrecto	Programador
Ejecutar pruebas de aceptación para cada caso de uso	Reporte de pruebas de aceptación	• Utilizar los casos de prueba de aceptación • Ejecutar el modulo de un caso de uso • Modificar código fuente si la prueba de aceptación muestra resultado incorrecto	Programador Usuario Cliente

Tabla N° 3.13: Implementación

Cuarto hito: liberación

3.5.5 TECNICA PARA APLICAR XP

Revisado el marco teórico desarrollado en el capitulo II, sección 2.2.3, formulamos el proceso, considerando las fases para desarrollar el software aplicando XP, como se muestra en las tablas 3.14 a 3.16.

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Escribir historias de usuario	Historia de usuario	• Describir brevemente la historia de usuario con la regla del negocio (lo que el sistema debe hacer) • Dividir historias de usuario grandes	Cliente
Probar las tecnologías a utilizar	Arquitectura técnica inicial	• Explorar posibilidades de uso de tecnologías • Probar el rendimiento de las tecnologías • Definir las tecnologías a usar	Cliente Programador Entrenador
Estimar esfuerzo para historias de usuario	Plan de alto nivel	• Conocer previamente la historia de usuario • Hacer una implementación rápida de historia de usuario • Estimar esfuerzo (semana) para desarrollar la historia de usuario	Programador

Tabla N° 3.14: Exploración

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Rescribir las historias de usuario	Historia de usuario	• Describir detalladamente la historia de usuario con la regla del negocio	Cliente

Formular el plan de versiones	Plan de versión (una iteración)	• Introducir nuevos requisitos del software • Definir prioridad para cada historia de usuario por necesidad del negocio	Cliente
		• Utilizar técnicas de elaboración del plan de alto nivel • Estimar y asignar esfuerzo (semana) para cada historia de usuario en función a tiempo para planear, diseñar, implementar y probar • Estimar y asignar riesgo a cada historia de usuario en función a situación que afecta la estimación del esfuerzo • Actualizar tarjeta de historia de usuario	Programador

Tabla Nº 3.15: Planificación

TAREA	ARTEFACTO	TÉCNICA	RESPONSABLES
Definir la arquitectura técnica	Arquitectura técnica	• Actualizar la arquitectura técnica inicial • Usar características del negocio • Utilizar arquitectura por capas • Integrar frameworks	Cliente Programador Entrenador
Escribir tareas de ingeniería	Tarea de ingeniería	• Dividir cada historia de usuario en tareas, describir usando reglas del negocio cada tarea de ingeniería	Cliente Programador
Formular el plan de iteraciones	Plan de iteración	• Estimar y asignar esfuerzo para desarrollar una tarea de ingeniería	Programador
		• Asignar una tarea de ingeniería al programador	Entrenador Programador
		• Utilizar el plan de versión • Actualizar el plan con tareas de ingeniería de la siguiente iteración • Actualizar el plan cuando fallo prueba de aceptación • Actualizar el plan con tareas no concluidas • Actualizar las tarjetas de tarea de ingeniería	Programador Entrenador Supervisar
Crear pruebas de aceptación	Caso de prueba de aceptación	• Escribir pruebas de aceptación para cada historia de usuario por iteración	Cliente Encargado de pruebas
Implementar las interfaces	GUI	• Diseñar con precisión la GUI relacionada a cada historia de usuario	Cliente Programador

		• Generar código para la interfase usando una herramienta	
Escribir tarjetas CRC para cada tarea de ingeniería	Tarjeta CRC	• Diseñar para una tarea de ingeniería de forma simple • Rediseñar por falla de prueba de aceptación una tarea • Identificar responsabilidades • Identificar colaboración • Identificar atributos	Cliente Programador
Implementar la base datos física	Base de datos física	• Escribir script usando tarjetas CRC • Ejecutar script usando DBMS	Programador
Implementar código para clases entidad	Código fuente	• Escribir código fuente o generar con una herramienta usando tarjetas CRC	Programador
Crear pruebas unitarias para las clases control	Prueba unitaria	• Escribir código fuente para una prueba unitaria, usando una herramienta	Programador
Implementar código fuente	Código fuente	• Codificar una tarea de ingeniería • Hacer refactoring • Mover programadores	Programador Supervisor
Ejecutar pruebas unitarias	Reporte de prueba unitaria	• Ejecutar el modulo de cada prueba unitaria • Modificar código fuente si la prueba unitaria muestra resultado incorrecto	Programador
Realizar integración continua	Código fuente	• Integrar las tareas para una historia de usuario • Mantener sistema integrado todo el tiempo	Programador
Ejecutar pruebas de integración para una historia de usuario	Reporte pruebas de integración	• Integrar continuamente al concluir las tareas de una historia de usuario • Verificar que las pruebas de integración pasan al 100%	Programador
Ejecutar pruebas de aceptación	Reporte de pruebas de aceptación	• Correr la ultima versión de una iteración • Utilizar los casos de prueba de aceptación	Cliente Encargado de pruebas

Tabla N° 3.16: Iteración

Cuarta fase: Producción

3.5.6 TECNICA PARA EVALUAR LA CALIDAD DEL PRODUCTO SOFTWARE

A. Ponderación por Característica

Peso atribuido a cada característica a evaluar, el peso es una base para la valoración por característica de acuerdo a la importancia que tiene

para el usuario. Asimismo, es el valor esperado por el usuario que la característica del producto software alcance luego de la evaluación, con un rango de cero a uno, ver tabla 3.17. La elección de la ponderación por característica y sub característica, se realizó mediante la recopilación de datos de las empresas Gestor Osmos SAC y Royal Systems SAC, quienes realizan aseguramiento de calidad de sus productos software, considerando el promedio de valores para cada característica y sub característica, valores establecidos por la experiencia de las empresas mencionadas para evaluar la calidad de sus productos software con características similares al producto desarrollado, asimismo, estas empresas incorporan dentro de la ponderación de cada característica y sub característica el valor sugerido por el usuario (cliente).

Ponderación_total_requerida = ∑ (Peso_caracteristica*Nivel_requerido)

CARACTERISTICA	PESO CARACTERISTICA	NIVEL REQUERIDO
Funcionalidad	30.00%	77.00%
Fiabilidad	10.00%	67.00%
Usabilidad	30.00%	74.00%
Eficiencia	10.00%	54.00%
Facilidad de Mantenimiento	10.00%	66.00%
Portabilidad	10.00%	66.00%
	Ponderación total requerida	**70.60%**

Tabla N° 3.17: Ponderación por característica

B. Ponderación por Sub Característica

Peso atribuido a cada sub característica a evaluar, el peso y el nivel requerido son una base para la valoración por sub característica de acuerdo a la importancia que tiene para el usuario. Asimismo, es el valor esperado por el usuario que la sub característica del producto software alcance luego de la evaluación con un rango de cero a uno, ver tabla 3.18.

Ponderación_por_caracteristica = ∑ (Peso_subcaracteristica*Nivel_requerido)

CARACTERISTICA	SUB CARACTERISTICA	PESO SUB CARACTERISTICA	NIVEL REQUERIDO
Funcionalidad	Aplicabilidad	20.00%	70.00%
	Precisión	35.00%	80.00%

	Interoperatividad	20.00%	75.00%
	Seguridad	25.00%	80.00%
		Ponderación por característica requerido	**77.00%**

Tabla N° 3.18: Ponderación por sub característica

C. Calificación

Valor de calidad alcanzado por el producto software, luego de evaluar aplicando métricas internas y externas, su rango va de cero a uno.

C.1 Calificación por Atributo

Valor obtenido para cada atributo evaluado, relacionado a una sub característica, ver tabla 3.19.

PRECISION			
NOMBRE	**PROPOSITO**	**MEDICION/FORMULA**	**NIVEL OBTENIDO**
exactitud de cálculos (métrica interna)	¿Cuán completamente se implementaron los requerimientos de exactitud?	X = A/B A = Número de funciones que se han implementado requerimientos de exactitud específicos, confirmados en la evaluación. B = Número de funciones para las cuales se necesita implementar requerimientos de exactitud específicos	70.00%
exactitud de cálculos (métrica externa)	¿Cuan frecuente los usuarios finales encuentran resultados inexactos?	X = A/T A = Número de cálculos inexactos encontrados por los usuarios. T = Tiempo de operación	50.00%

Tabla N° 3.19: Calificación por sub característica – atributos

C.2 Calificación Individual por Sub Característica

Valor calculado, es el promedio del nivel obtenido de cada atributo, para cada sub característica, ver tabla 3.20.

Ponderacion_subcaracteristica = Promedio(Nivel_obtenido_cada_atributo)

CARACTERÍSTICA	SUB CARACTERÍSTICA	PESO SUB CARACTERISTICA	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	Aplicabilidad	20.00%	70.00%	80.00%
	Precisión	35.00%	80.00%	**60.00%**
	Interoperatividad	20.00%	75.00%	70.00%
	Seguridad	25.00%	80.00%	60.00%

Tabla N° 3.20: Calificación individual por sub característica

C.3 Calificación Ponderada por Característica

Valor calculado ponderado para cada característica, ver la tabla 3.21.

Ponderación_característica_obtenido = ∑ (Peso_subcaracteristica*Nivel_obtenido)

CARACTERISTICA	SUB CARACTERISTICA	PESO SUB CARACTERISTCA	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	Aplicabilidad	20.00%	70.00%	80.00%
	Precisión	35.00%	80.00%	60.00%
	Interoperatividad	20.00%	75.00%	70.00%
	Seguridad	25.00%	80.00%	60.00%
			Ponderación por característica obtenido	**66.00%**

Tabla N° 3.21: Calificación ponderada por característica

C.4 Calificación Ponderada Total

Es el valor calculado mediante el peso de cada característica por el nivel obtenido, encontrado la ponderación total obtenida, ver tabla 3.22.

Ponderación_total_obtenida = ∑ (Peso_caracteristica*Nivel_obtenido)

CARACTERISTICA	PESO CARACTERISTICA	NIVEL OBTENIDO
Funcionalidad	30.00%	66.00%
Fiabilidad	10.00%	69.50%
Usabilidad	30.00%	69.80%
Eficiencia	10.00%	70.48%
Facilidad de Mantenimiento	10.00%	64.48%
Portabilidad	10.00%	75.00%
	Ponderación total obtenida	**68.69%**

Tabla N° 3.22: Calificación ponderada total

C.5 Calificación Final

Valoración del control de calidad del producto software. Mediante la diferencia entre nivel requerido (ponderación total requerida) y nivel obtenido (ponderación total obtenida), se establece que características cumplen con el nivel requerido, como se muestra en la tabla 3.23.

CARACTERISTICA	RESULTADO	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	No Cumple	77.00%	66.00%
Fiabilidad	Cumple	67.00%	69.50%
Usabilidad	No Cumple	74.00%	69.80%

Eficiencia	Cumple	54.00%	70.48%
Facilidad de Mantenimiento	No Cumple	66.00%	64.48%
Portabilidad	Cumple	66.00%	75.00%
	Total	**70.60%**	**68.69%**

Tabla N° 3.23: Calificación final

D. Matriz Base para Evaluar la Calidad del Producto Software

La tabla 3.24 presenta los pesos, el nivel requerido y nivel obtenido para las características y sub características de acuerdo al modelo de calidad NTP-ISO/IEC 9126-1:2004.

CARACTERISTICA	PESO CARAC-TERISTICA	SUB CARAC-TERISTICA	PESO SUB CARAC-TERISTICA	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	20.00%	Aplicabilidad	20.00%	70.00%	80.00%
		Precisión	35.00%	80.00%	60.00%
		Interoperatividad	20.00%	75.00%	70.00%
		Seguridad	25.00%	80.00%	60.00%
			100.00%	**77.00%**	**66.00%**
Fiabilidad	30.00%	Madurez	35.00%	80.00%	60.00%
		Tolerancia a fallas	30.00%	60.00%	80.00%
		Recuperabilidad	35.00%	60.00%	70.00%
			100.00%	**67.00%**	**69.50%**
Usabilidad	5.00%	Entendibilidad	35.00%	80.00%	76.00%
		Facilidad de aprendizaje	30.00%	80.00%	64.00%
		Operabilidad	30.00%	65.00%	66.00%
		Atractividad	15.00%	50.00%	70.00%
			100.00%	**74.00%**	**69.80%**
Eficiencia	5.00%	Comportamiento en el tiempo	60.00%	50.00%	62.40%
		Utilización de recursos	40.00%	60.00%	82.60%
			100.00%	**54.00%**	**70.48%**
Facilidad de Mantenimiento	30.00%	Analizabilidad	15.00%	60.00%	52.20%
		Cambiabilidad	25.00%	70.00%	62.80%
		Estabilidad	25.00%	60.00%	65.00%
		Testeabilidad	35.00%	70.00%	74.00%
			100.00%	**66.00%**	**64.48%**
Portabilidad	10.00%	Adaptabilidad	30.00%	80.00%	75.00%
		Instalabilidad	25.00%	60.00%	72.00%
		Coexistencia	30.00%	60.00%	80.00%
		Reemplazabilidad	15.00%	60.00%	70.00%
	100.00%		**100.00%**	**66.00%**	**75.00%**
			TOTAL	**70.60%**	**68.69%**

Tabla N° 3.24: Base para evaluar la calidad del producto software

E. Interpretación del Resultado

Valorar la calidad del producto software desde la óptica del desarrollador es interpretar el grado de calidad, para lo cual se construye primero una escala: deficiente, insuficiente, satisfactorio y excede exigencia. En base a encuestas realizadas al personal de desarrollo, ver tabla 3.25.

RESULTADO	ESCALA
25%	Deficiente
60%	Insuficiente
80%	Satisfactorio
100%	Excede exigencia

Tabla N° 3.25: Escala para valorar la calidad del producto software

Luego construir una escala por rangos de porcentajes usando las tablas 3.24 y 3.25. Ver la tabla 3.26, construida de la siguiente forma:

Deficiente = (Nivel_requerido_total/Nivel_obtenido_total)*(%_deficiente/100)
Insuficiente = Nivel_requerido_total
Satisfactorio = (Nivel_requerido_total/Nivel_obtenido_total)*(%_satisfactorio/100)

RANGO	ESCALA PARA VALORAR
[0 - Deficiente>	Deficiente
[Deficiente - Insuficiente>	Insuficiente
[Insuficiente - Satisfactorio>	Satisfactorio
[Satisfactorio - 100]	Excede exigencia

Tabla N° 3.26: Rangos para interpretar la calidad del producto software

3.5.7 ANALISIS ESTADISTICO

Las mediciones obtenidas de la calidad del producto presentan medición cuantitativa proporcional, entonces utilizamos la técnica estadística Ji-cuadrado, para distribución multinomial, la cual nos permitirá definir cual de las metodologías (ICONIX o XP), produce software con mejor calidad de producto. Primero, formulamos la hipótesis nula (Ho), donde las diversas proporciones asociadas a las características en evaluación son iguales y, la hipótesis alterna (H1) donde al menos una de las proporciones es diferente.

Luego, procesamos el análisis de diferencias de proporciones mediante la prueba Ji-cuadrada para establecer la validez de los supuestos, mediante el procedimiento siguiente:

a. Probar que cada conjunto de datos es independiente del resto;

b. Determinar si cada grupo (resultados obtenidos para la calidad del producto aplicando ICONIX y XP) sigue una distribución multinomial.

c. El supuesto principal para usar el estadístico de prueba Ji-cuadrado es que el tamaño muestral sea superior a 30 datos, cumplido ello procederemos a ejecutar la prueba de igualdad de proporciones múltiples y, mediante el estadístico χ^2 obtenemos el p-valor. Es decir obtener el verdadero valor de cometer un error tipo I (rechazar la Ho como falsa, cuando en realidad Ho es verdadera). Entonces, si el p-valor es menor al 1% rechazamos Ho. En caso contrario, si el p-valor es igual o mayor al 1% no podemos rechazar Ho. Donde, el nivel de significación (α) es decir el riesgo que se asume al rechazar la Ho, se eligió $\alpha = 0.01$, por ser este el valor óptimo en el control de calidad.

d. Realizado la validez de los supuestos, procederemos a ejecutar la prueba Ji-cuadrado. Esta es una técnica estadística que nos permite comparar más de dos proporciones al mismo tiempo comparándolas con proporciones multinomiales teóricas, previamente establecidas. El estadístico de prueba se distribuye con una Ji-cuadrado con k-1 grados de libertad, el estadístico de prueba esta dado por:

$$\chi^2 = \sum_{i=1}^{K} (Oi - Ei)^2 / Ei$$

Donde:

Oi, viene a ser los valores observados de la muestra.

Ei, viene a ser los valores esperados de la hipótesis nula.

e. En esta técnica estadística y para nuestra investigación, la hipótesis nula (Ho) planteada es que las proporciones de evaluación del nivel obtenido, es igual a las proporciones de evaluación del nivel requerido.

La hipótesis alternativa (Ha) planteada es que las proporciones de evaluación del nivel obtenido es superior a las proporciones de evaluación del nivel requerido.

Para probar la igualdad de proporciones asociadas a las sub-características, utilizaremos la prueba Z como aproximación binomial. Dado que en numerosas ocasiones se plantea estimar una proporción o porcentaje. En estos casos la variable aleatoria toma solamente dos valores diferentes (éxito o fracaso), es decir sigue una distribución binomial y cuando la extensión de la población es grande la distribución binomial B(n,p) se aproxima a la normal $N(np, \sqrt{npq})$.

Para muestras de tamaño n>30, la distribución muestral de proporciones sigue una distribución normal.

$$N(p, \sqrt{\frac{pq}{n}})$$

Donde;
p, es la proporción de uno de los valores que presenta la variable estadística en la población y q=1-p.

CAPITULO IV

ANALISIS Y RESULTADOS DE LA INVESTIGACION

4.1 RESULTADOS

4.1.1 ARTEFACTOS DEL SOFTWARE APLICANDO EL PROCESO ICONIX

Aplicando las técnicas de la metodología ICONIX resumidas en las tablas 3.6 a 3.13, obtenemos los artefactos como tablas y figuras para el modelado e implementación del software para comercializar tara en la Región Ayacucho.

A. Análisis de Requisitos

De acuerdo a la figura 2.31 e información obtenida mediante el anexo E, usando las técnicas de la tabla 3.6, obtenemos los artefactos siguientes:

Nº Req.	Requisitos
	FUNCIONALES
01	El **administrador** debe ser capaz de crear una **cuenta** para un **usuario**, previa **solicitud** de acceso
02	El software debe ser capaz de proveer la actualización de las **características** de la **tara** y sus **derivados**
03	El **productor** debe ser capaz de publicar su **oferta** para la **venta** de tara
04	El productor puede registrar una **boleta**, **factura** por la venta de tara
05	El productor debe ser capaz de realizar una **proforma** ante la **cotización** de un **acopiador** o **transformador** para la venta de tara
06	El productor debe ser capaz de realizar una **cotización** de **bien** o **servicio** a un **proveedor** (**actor indirecto**) para comercializar tara
07	El acopiador debe ser capaz de publicar su oferta para la venta de tara
08	El acopiador podrá registrar una boleta, factura o contrato por la venta de tara
09	El acopiador podrá ser capaz de emitir una proforma ante la cotización de un transformador u otro **acopiador** para la venta de tara
10	El acopiador puede realizar una cotización de bien o servicio a un proveedor para comercializar tara
11	El acopiador puede realizar una cotización de tara al productor para la compra
12	El transformador debe ser capaz de publicar su oferta para la venta de tara y su derivados
13	El transformador podrá registrar una boleta, factura o contrato por la

	venta de tara y sus derivados
14	El transformador podrá emitir una proforma ante la cotización de un **cliente** para la venta de tara y sus derivados
15	El transformador será capaz de realizar una cotización de bien o servicio para comercializar tara
16	El transformador puede realizar una cotización de tara y sus derivados al acopiador o productor para la compra
17	El cliente puede ser capaz de realizar una cotización de tara y sus derivados al transformador
18	El software debe ser capaz de proveer la actualización de las características de un bien o servicio
19	El proveedor de bien o servicio debe ser capaz de publicar su oferta de un bien o servicio
20	El proveedor de bien o servicio puede realizar una proforma ante la cotización de un cliente (**actor directo**) para la venta de bien o servicio
21	El proveedor de bien o servicio (actor indirecto) debe ser capaz de registrar una boleta o factura por la venta de un bien o servicio para comercializar tara
22	El software debe ser capaz de mostrar la matriz de **competencia** para la tara y su derivados de la **región**
23	El software debe ser capaz de mostrar la **demanda** y los **mercados** para la tara y sus derivados
24	El software debe ser capaz de mostrar la matriz histórica de precios para la tara y sus derivados
	NO FUNCIONALES
25	El software para comercializar tara debe ser una aplicación web y contar con ayudas para recordar la clave de acceso.
26	El software debe tener las ayudas necesarias para su aprendizaje y correcta operación.
27	El software debe ser capaz de ejecutarse en cualquier sistema operativo garantizando su portabilidad
28	El software debe presentar interfaces fáciles de utilizar
29	El software debe ser personalizable para garantizar el cumplimiento del rol de un actor
30	El software debe presentar una arquitectura técnica y codificación usando estándares que permita su operación y mantenimiento adecuado

Tabla N° 4.1: Requisitos funcionales y no funcionales.

N° Req.	N° C.P	Casos de Prueba de Aceptación
05 09 14	01	Comprobar que los datos del actor (persona natural o jurídica) proveedor del producto tara son correctos.
	02	Comprobar que la cotización, es de un actor (transformador, acopiador y cliente) y presenta información correcta sobre la fecha de vigencia, descripción, unidad, cantidad.
	03	Comprobar que el producto tara, provee un actor (productor, transformador y acopiador) y presenta código, descripción y características correctas.
	04	Verificar que la cantidad, el precio unitario, descuento, subtotal y total para un ítem son correctos.
	05	Imprimir la proforma de productos tara y verificar que los datos se grabaron correctamente.
04 08	01	Comprobar que los datos del actor proveedor de productos tara son correctos.

13	02	Comprobar que los datos del actor cliente seleccionado (transformador, acopiador y cliente) son correctos.
	03	Verificar que el producto tara vendido, pertenece a un actor proveedor (productor, transformador y acopiador) y presenta código, descripción y características correctas.
	04	Verificar que la cantidad, el precio unitario, descuento, subtotal y total para un ítem son correctos.
	05	Imprimir el comprobante de pago de los productos tara vendidos y verificar que los datos se grabaron correctamente.

Tabla N° 4.2: Casos de prueba de aceptación. Dos casos de uso

Glosario de términos

Administrador	Derivado	Proforma	Mercado
Cuenta	Venta	Demanda	Acopiador
Usuario	Transformador	Productor	Cotización
Solicitud	Cliente	Oferta	Bien
Característica	Actor directo	Servicio	Boleta
Tara	Competencia	Actor indirecto	Factura

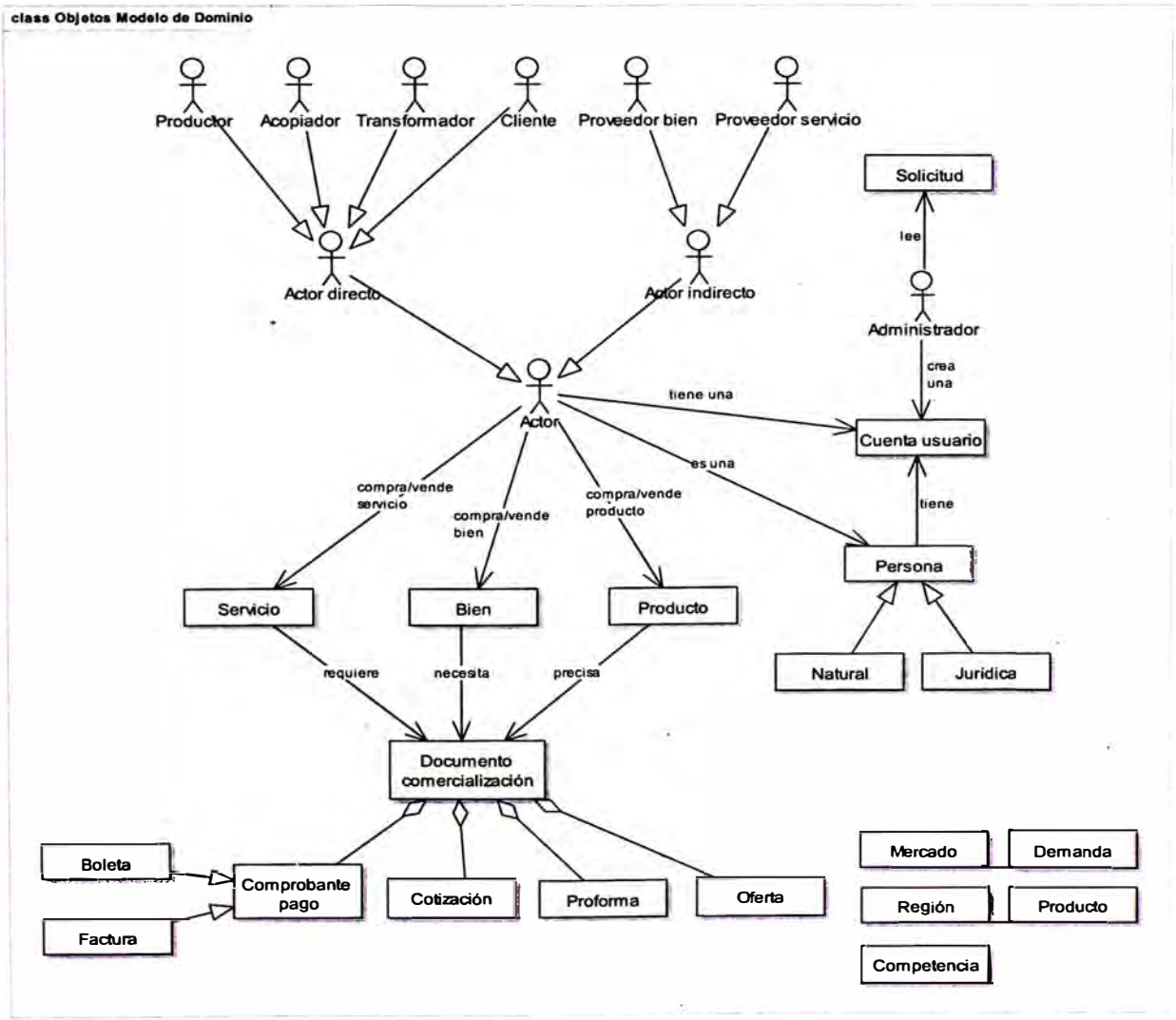

Figura N° 4.1: Modelo de dominio.

PROFORMA PRODUCTO

---------- Proveedor ----------

Nombre o R/S: ..... Actor: ..... RUC: .....

Dirección: ..... Fecha Emisión: ..... N° Proforma: .....

---------- Cliente ----------

Nombre o R/S: ..... Actor: ..... RUC: .....

Dirección: ..... Localidad: .....

Item	Descripción	Unidad	Cantidad	Precio Unitario (S/.)	Sub Total (S/.)

Fecha Vigencia: .....

TOTAL (S/.) .....

Figura N° 4.2: Prototipo de interfaz grafica básica "Emitir proforma producto tara".

COMPROBANTE DE PAGO

---------- Proveedor ----------

Nombre o R/S: ..... Actor: ..... RUC: .....

Dirección: ..... Fecha Emisión: ..... N° C/P: .....

---------- Cliente ----------

Nombre o R/S: ..... Actor: ..... RUC: .....

Dirección: ..... Localidad: .....

Item	Descripción	Unidad	Cantidad	Precio Unitario (S/.)	Sub Total (S/.)

IGV(S/.) .....

Fecha Cancelación: .....

TOTAL (S/.) .....

Figura N° 4.3: Prototipo de interfaz grafica básica "Registrar comprobante pago producto tara".

N° C.U	Casos de Uso
01	Emitir proforma producto tara
02	Mantener proforma producto tara
03	Emitir proforma bien
04	Mantener proforma bien
05	Emitir proforma servicio
06	Mantener proforma servicio
07	Registrar comprobante pago producto tara
08	Mantener comprobante pago producto tara
09	Registrar comprobante pago bien
10	Mantener comprobante pago bien
11	Registrar comprobante pago servicio
12	Mantener comprobante pago servicio

13	Emitir cotización producto tara
14	Mantener cotización producto tara
15	Emitir cotización bien
16	Mantener cotización bien
17	Emitir cotización servicio
18	Mantener cotización servicio
19	Publicar oferta producto tara
20	Mantener oferta producto tara
21	Publicar oferta bien
22	Mantener oferta bien
23	Publicar oferta servicio
24	Mantener oferta servicio
25	Emitir reporte competencia
26	Emitir reporte demanda
27	Emitir reporte precios históricos
28	Solicitar acceso como actor directo o indirecto
29	Mantener cuenta y personalizar actor directo o indirecto
30	Actualizar características producto tara
31	Actualizar características bien
32	Actualizar características servicio
33	Actualizar características producción

Tabla N° 4.3: Lista de casos de uso.

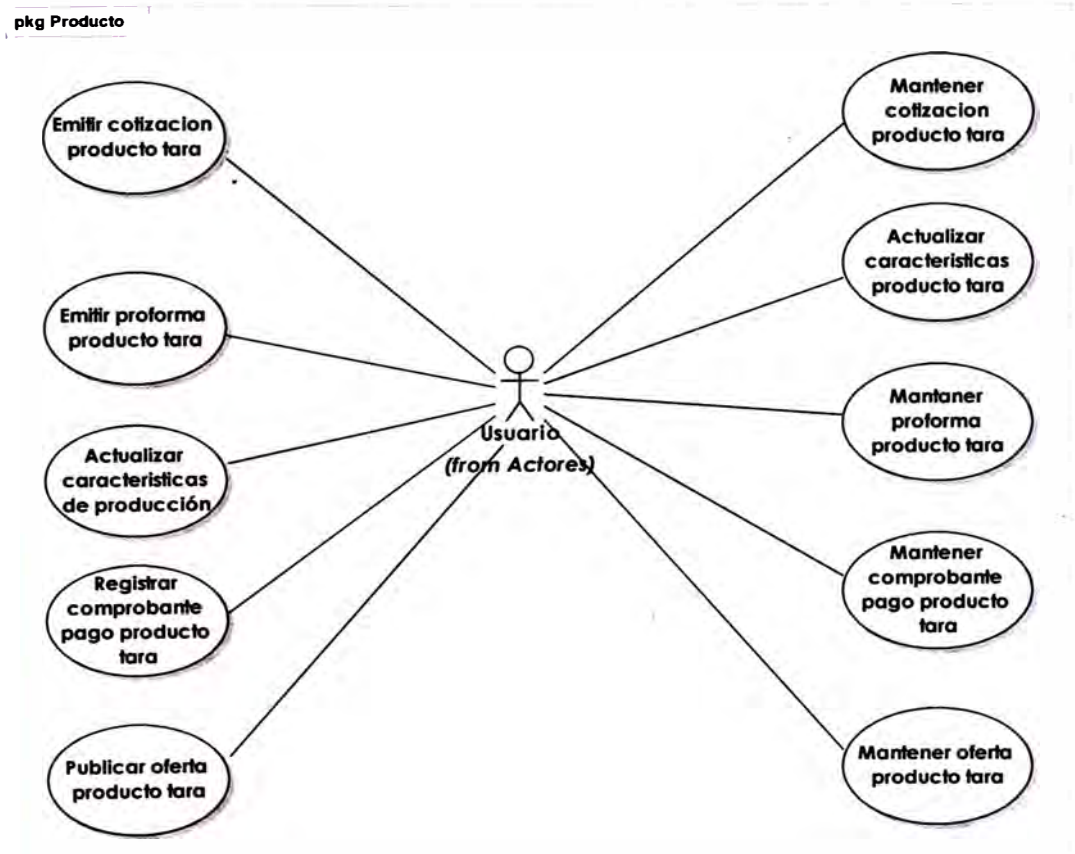

Figura N° 4.4: Casos de uso del paquete "Producto".

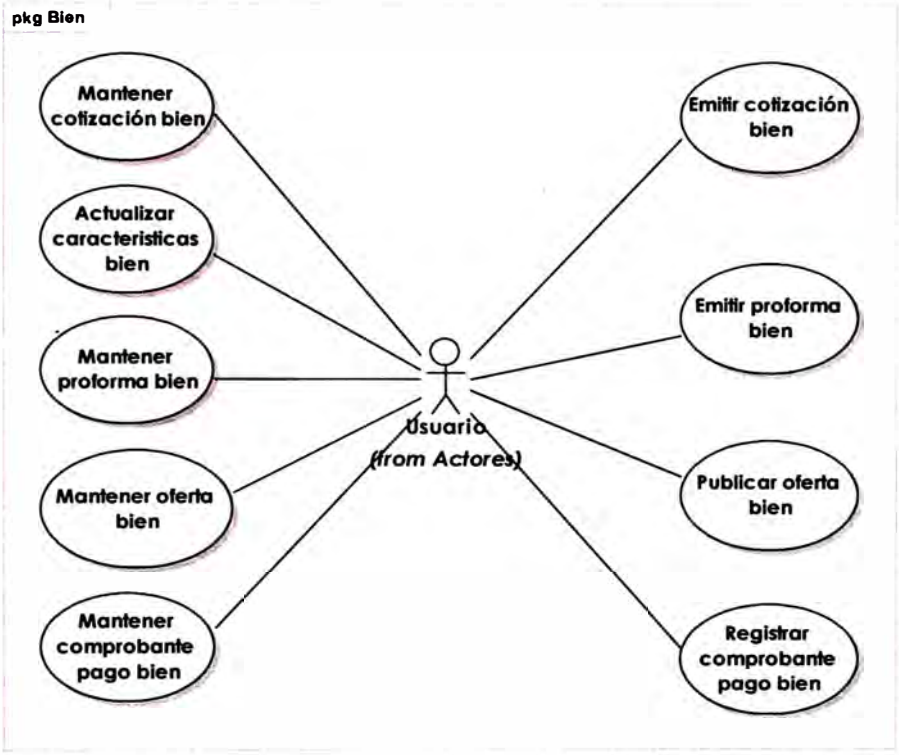

Figura N° 4.5: Casos de uso del paquete "Bien".

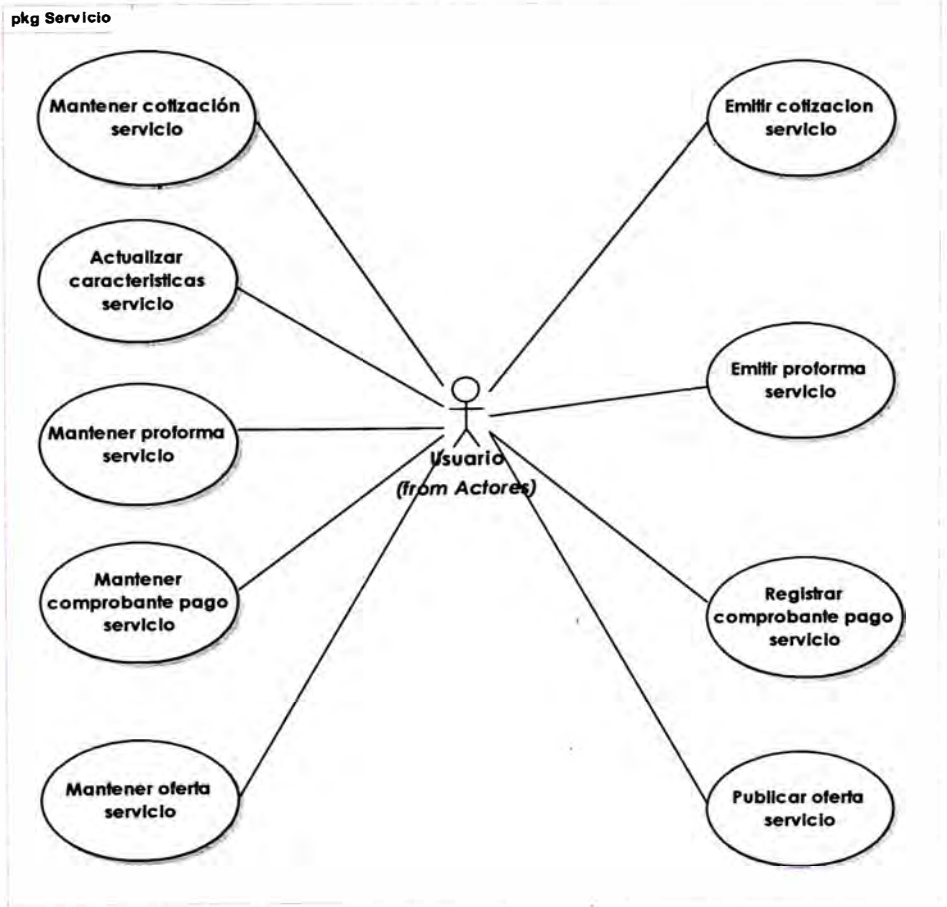

Figura N° 4.6: Casos de uso del paquete "Servicio".

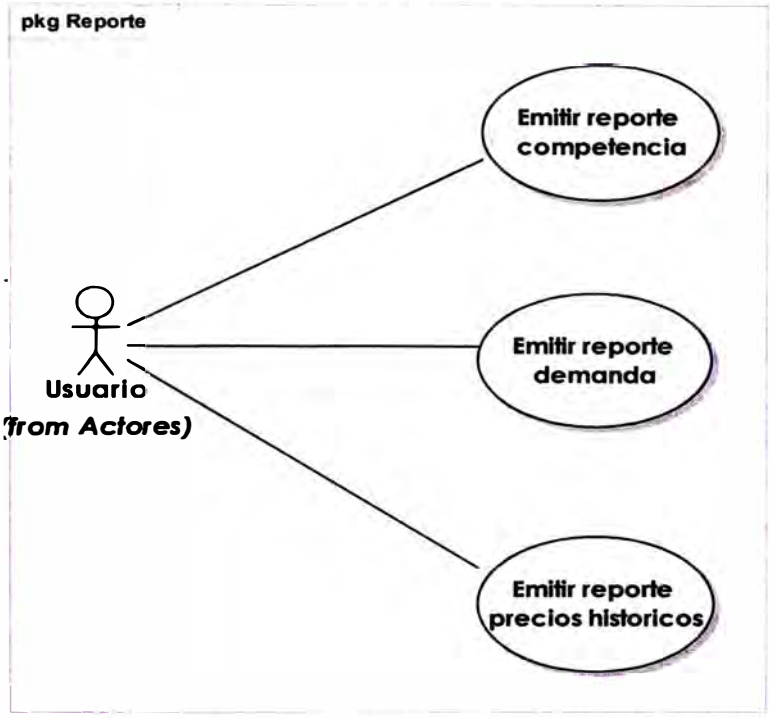

Figura N° 4.7: Casos de uso del paquete "Reporte".

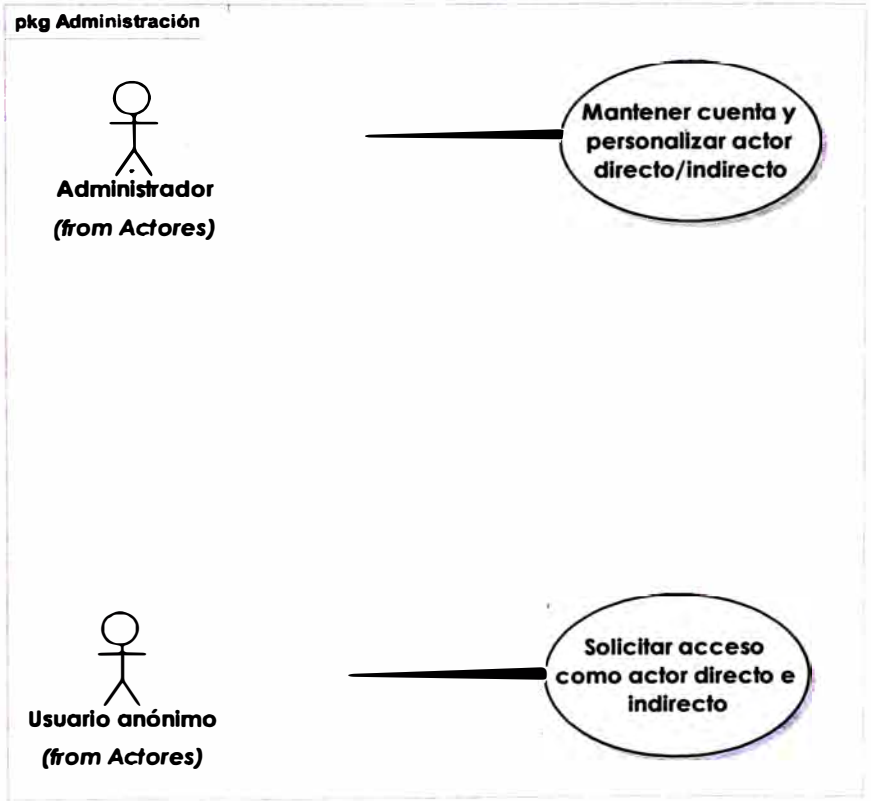

Figura N° 4.8: Casos de uso del paquete "Administración".

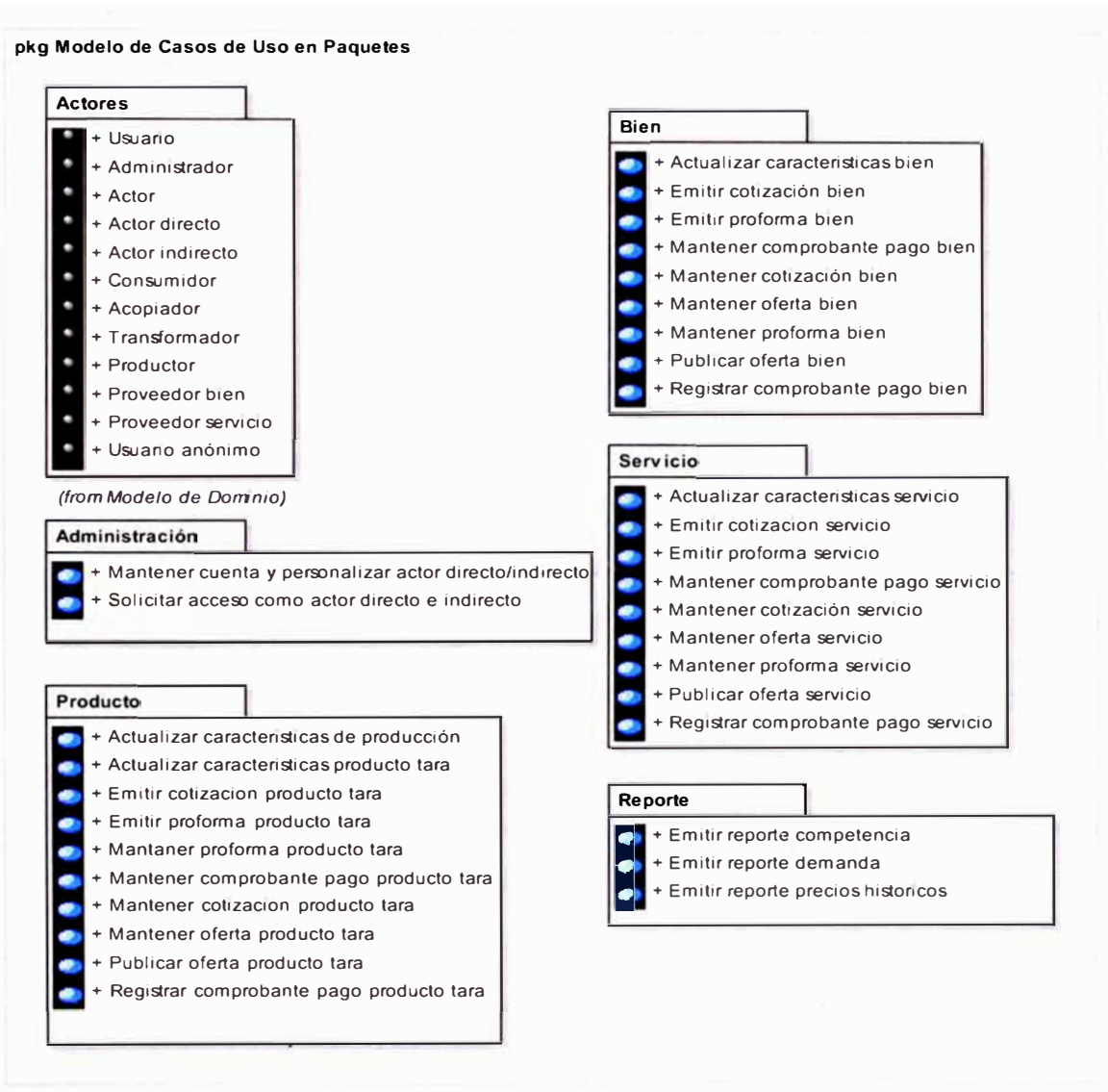

Figura N° 4.9: Casos de uso organizado por paquetes.

Requisitos Funcionales	Casos de Uso
Req 01. Crear cuenta usuario Req 26. El sistema debe ser capaz de imprimir la lista de usuarios por actor	CU 28. Solicitar acceso como actor directo o indirecto CU 29. Crear cuenta y personalizar actor directo o indirecto
Req 02. Actualizar características de producto tara	CU 30. Actualizar características producto tara
Req 03. Productor publica oferta venta producto tara Req 07. Acopiador publica oferta venta producto tara Req 12. Transformador publica oferta venta producto tara Req 27. El sistema debe emitir un reporte por mes, año y provincia de la oferta de producto tara	CU 19. Publicar oferta producto tara CU 20. Mantener oferta producto tara
Req 04. Productor registra comprobante venta producto tara	CU 07. Registrar comprobante pago producto tara

Req 08. Acopiador registra comprobante venta producto tara Req 13. Transformador registra comprobante venta producto	CU 08. Mantener comprobante pago producto tara
Req 05. Productor emite proforma por cotización Req 09. Acopiador emite proforma por cotización Req 14. Transformador emite proforma por cotización	CU 01. Emitir proforma producto tara CU 02. Mantener proforma producto tara
Req 06. Productor realiza cotización para bien o servicio Req 10. Acopiador realiza cotización para bien o servicio Req 15. Transformador realiza cotización para bien o servicio	CU 15. Emitir cotización bien CU 16. Mantener cotización bien CU 17. Emitir cotización servicio CU 18. Mantener cotización servicio
Req 11. Acopiador realiza cotización al productor Req 16. Transformador realiza cotización al acopiador o productor Req 17. Cliente realiza cotización de producto tara	CU 13. Emitir cotización producto tara CU 14. Mantener cotización producto tara
Req 18. Actualizar características de un bien o servicio	CU 31. Actualizar características bien CU 32. Actualizar características servicio
Req 19. Proveedor publica oferta bien y servicio	CU 21. Publicar oferta bien CU 22. Mantener oferta bien CU 23. Publicar oferta servicio CU 24. Mantener oferta servicio
Req 20. Proveedor emite proforma por cotización de bien y servicio	CU 03. Emitir proforma bien CU 04. Mantener proforma bien CU 05. Emitir proforma servicio CU 06. Mantener proforma servicio
Req 21. Proveedor registra comprobante venta bien o servicio	CU 09. Registrar comprobante pago bien CU 10. Mantener comprobante pago bien CU 11. Registrar comprobante pago servicio CU 12. Mantener comprobante pago servicio
Req 22. Mostrar reporte de competencia	CU 25. Emitir reporte competencia
Req 23. El sistema debe emitir un reporte por mes, año y provincia de la demanda del producto tara	CU 26. Emitir reporte demanda
Req 24. El sistema debe emitir el reporte de la variación de precios promedios históricos del producto tara por mes y año	CU 27. Emitir reporte precios históricos
Req 25. El productor registra las características de producción	CU 33. Actualizar características producción

Tabla N° 4.4: Relación entre requisitos funcionales y casos de uso.

De acuerdo a la metodología, a partir de la relación entre casos de uso y requisitos funcionales, elegimos un conjunto mínimo de casos de uso para

continuar el modelado (análisis y diseño), implementación y pruebas. Aquí, seleccionamos los casos de uso "Emitir proforma producto tara" y "Registrar comprobante pago producto tara", por considerarlos casos de uso eje en el desarrollo y mostramos sus artefactos de acuerdo a las fases e hitos, el resto de artefactos para los otros casos de uso se presentan en el anexo A.

Casos de Uso	Descripción
CU 01. Emitir proforma producto tara	**Curso básico:** El actor (productor, acopiador o transformador), hace clic en el menú "emitir proforma", el sistema muestra la pantalla emitir proforma producto e ingresa la fecha de emisión, número de proforma. El actor busca un cliente (acopiador, transformador o consumidor) y el sistema muestra los datos personales del cliente. El actor ingresa cada ítem (descripción, unidad, cantidad y precio unitario), el sistema muestra el sub total y el total. El actor ingresa fecha de vigencia y hace clic en botón "enviar proforma", el sistema graba la proforma. **Curso alterno** El sistema no graba la proforma y muestra un mensaje de error. El sistema muestra un mensaje de error cuando falta un dato.
CU 07. Registrar comprobante pago producto tara	**Curso básico** El actor (productor, acopiador o transformador), hace clic en el menú "registrar comprobante", el sistema muestra la pantalla registrar comprobante producto, con sus datos personales en la cabecera de la pantalla. El actor busca un cliente (acopiador, transformador o consumidor) y el sistema muestra los datos personales del cliente. El actor ingresa la fecha de emisión, número de comprobante de pago. El actor ingresa cada ítem (descripción, unidad, cantidad y precio unitario), el sistema muestra el sub total, IGV y total. El actor ingresa fecha de cancelación y hace clic en botón "registrar comprobante", el sistema graba el comprobante. **Curso alterno** El sistema no graba el comprobante y muestra un mensaje de error. El sistema muestra un mensaje de error cuando falta un dato.

Tabla N° 4.5: Primer borrador de casos de uso.

B. Revisión de Requisitos (primer hito)

Se revisa el modelo de dominio, se diseña la GUI final y se reescribe los borradores para los dos casos de uso de acuerdo a la tabla 3.7.

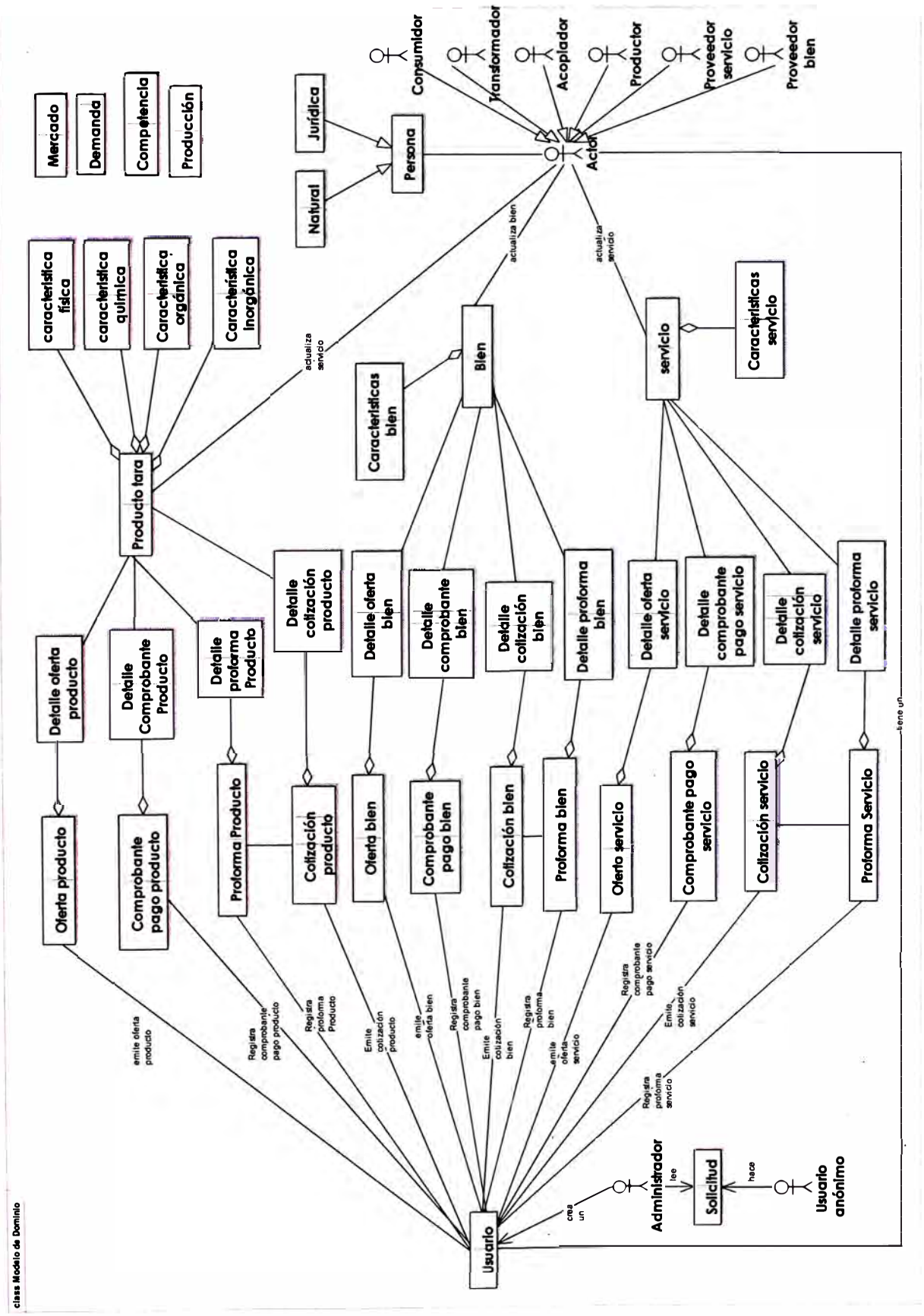

Figura N° 4.10: Modelo de dominio.

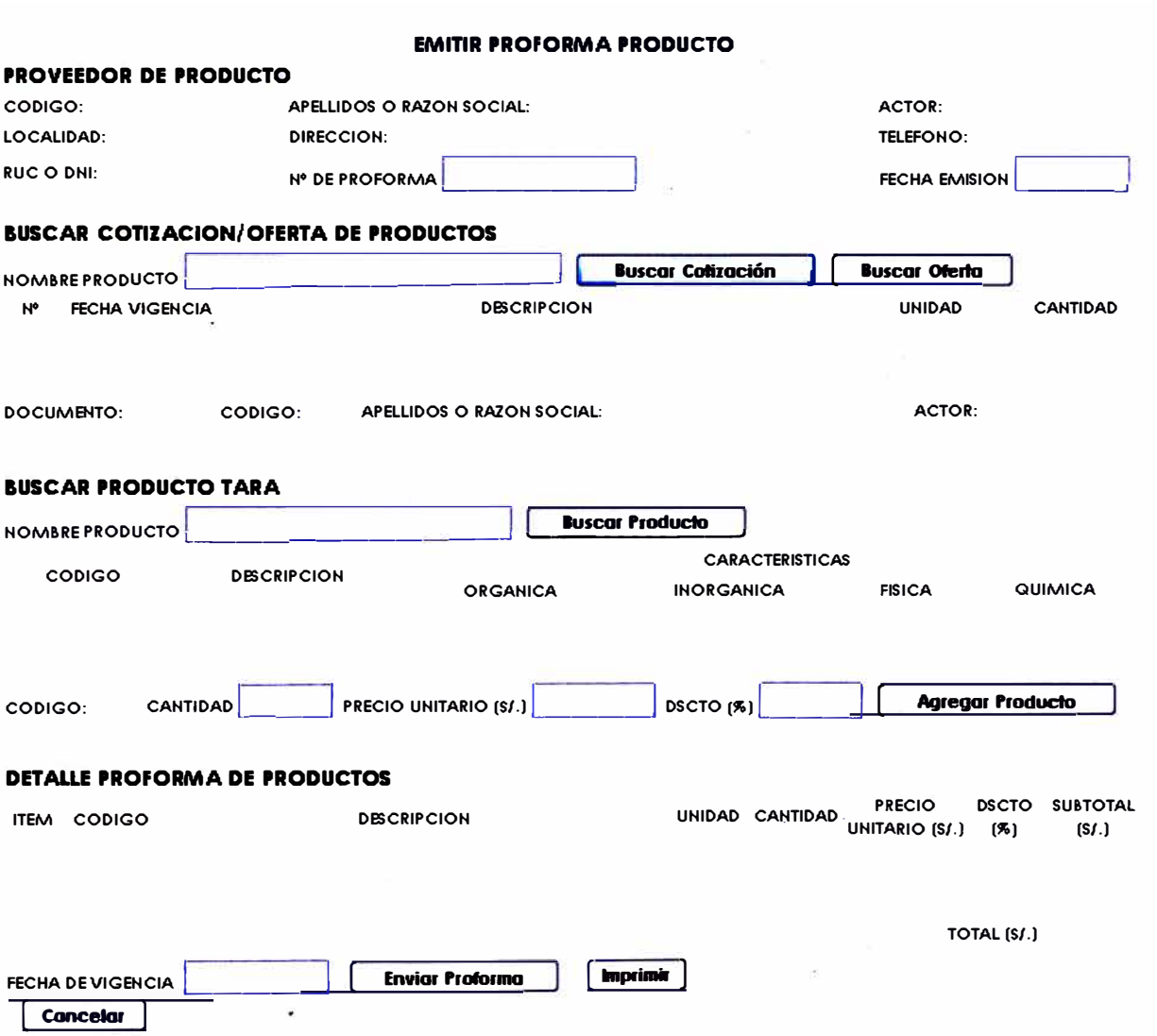

Figura Nº 4.11: Prototipo de interfaz grafica "Emitir proforma producto tara".

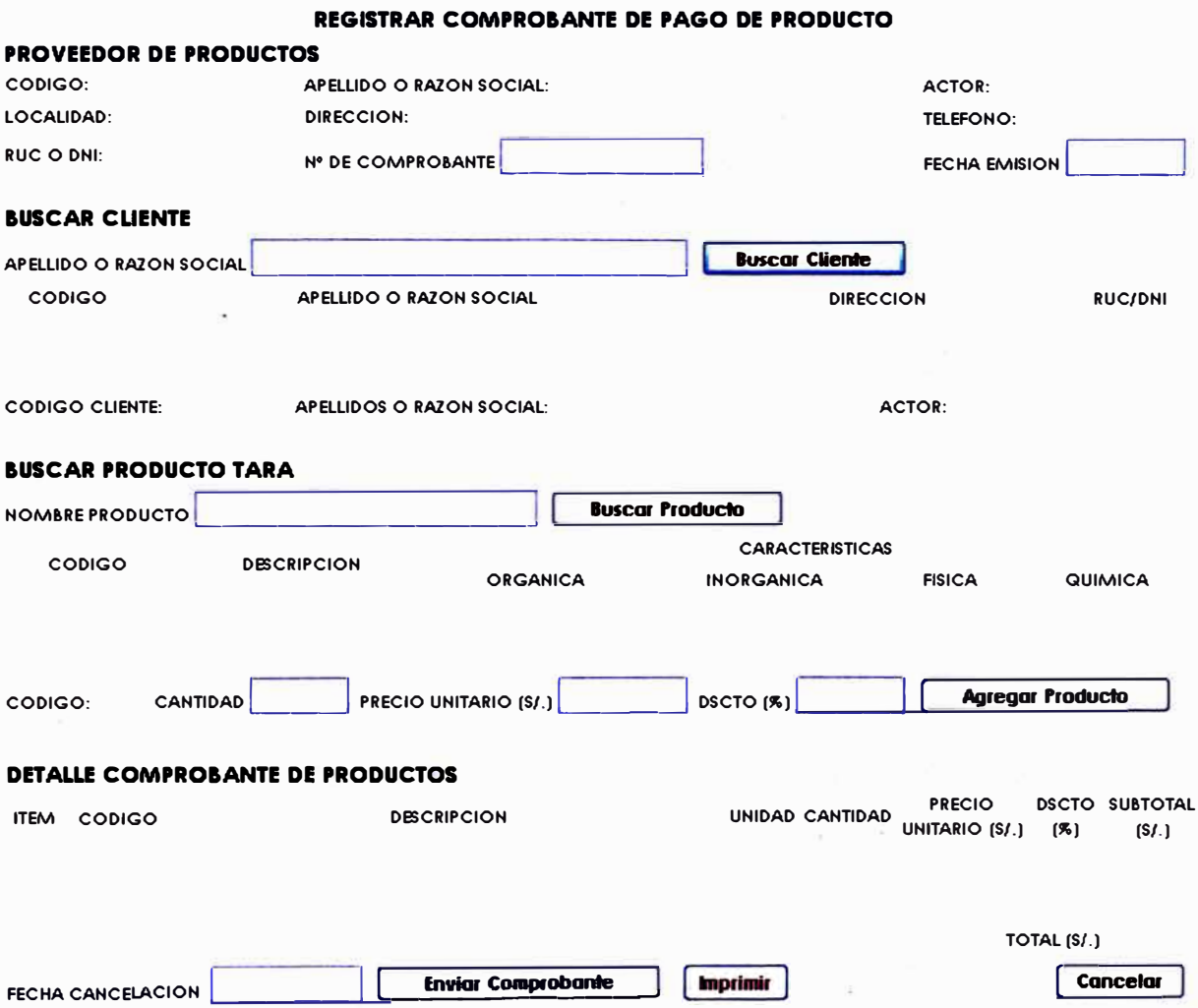

Figura Nº 4.12: Prototipo de interfaz grafica "Registrar comprobante pago producto tara".

Casos de Uso	Descripción
CU 01: Emitir proforma producto tara	**CURSO BÁSICO:** El usuario (productor, acopiador o transformador), hace clic en menú "emitir proforma", el sistema muestra la interfase "emitir proforma producto". El usuario ingresa el número de proforma, la fecha de emisión, nombre de producto tara y hace clic en buscar, el sistema muestra número de cotización y fecha de vigencia. El usuario hace clic en una cotización, el sistema muestra detalles de la cotización. El usuario ingresa nombre de producto tara y hace clic en buscar, el sistema muestra detalles de su producto. El usuario selecciona su producto, el sistema muestra el código de su producto seleccionado. El usuario ingresa cantidad, precio unitario, descuento y hace clic en agregar, el sistema muestra un ítem agregado a su proforma, calcula el subtotal, el IGV y total. El usuario ingresa fecha de vigencia y hace clic en enviar proforma, el sistema muestra mensaje de "proforma enviado". **CURSO ALTERNO:** **Nombre de producto no encontrado en cotizaciones:** El sistema muestra mensaje de error "nombre de producto no encontrado en cotizaciones". **Nombre de producto no encontrado en sus productos registrados:** El sistema muestra un mensaje de error "nombre de producto no encontrado". **Sistema no envía la proforma:** El sistema muestra mensaje de error "proforma no enviado".
CU 07: Registrar comprobante pago producto tara	**CURSO BÁSICO:** El usuario (productor, acopiador o transformador), hace clic en menú "Registrar comprobante producto", el sistema muestra la interfase "Registrar comprobante producto". El usuario ingresa el número de factura/boleta, la fecha de emisión, apellidos o razón social y hace clic en buscar, el sistema muestra código, apellidos o razón social. El usuario hace clic en cliente, el sistema muestra detalles del cliente. El usuario ingresa nombre de producto tara y hace clic en buscar, el sistema muestra detalles de su producto. El usuario selecciona su producto, el sistema muestra el código de su producto seleccionado. El usuario ingresa cantidad, precio unitario, descuento y hace clic en agregar, el sistema muestra un ítem agregado a su factura/boleta, calcula el subtotal, el IGV y total. El usuario ingresa fecha de cancelación y hace clic en enviar comprobante, el sistema muestra mensaje de "comprobante Registrado". **CURSO ALTERNO:** **Cliente no encontrado en la búsqueda:** El sistema muestra mensaje de error "nombre de Cliente no encontrado". **Nombre de producto no encontrado en sus productos registrados:** El sistema muestra un mensaje de error "nombre de producto no encontrado". **Sistema no envía el comprobante:** El sistema muestra mensaje de error "comprobante no enviado".

Tabla N° 4.6: Primer borrador de casos de uso.

C. Diseño Preliminar

Durante el diseño preliminar trabajamos con los dos casos de uso seleccionados y las técnicas para construir un diagrama de robustez, descritas en la tabla 3.8, el caso de uso desambiguado se muestra en cada diagrama de robustez y el modelo de dominio actualizado presenta atributos y clases para los dos casos de uso seleccionados.

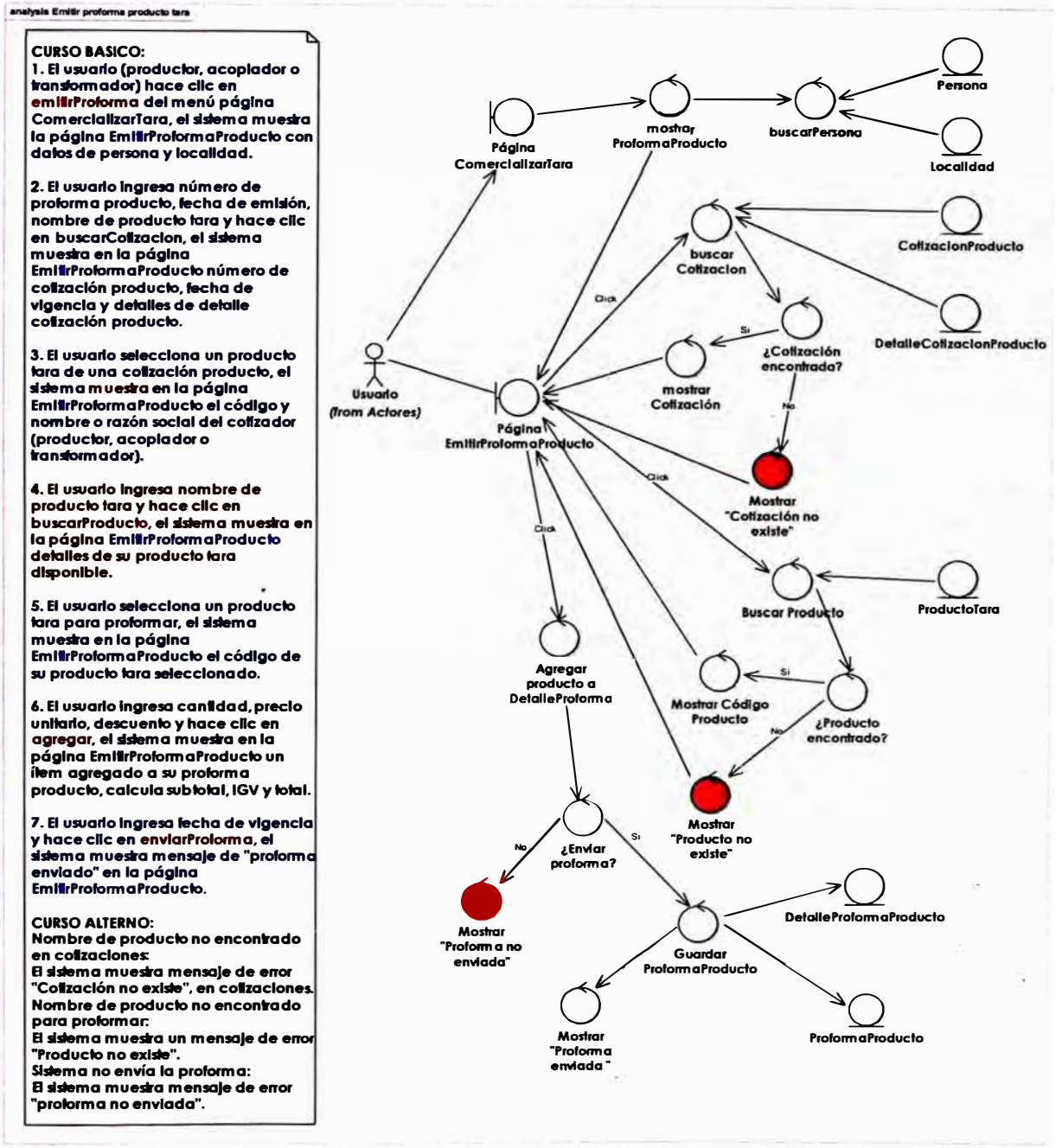

Figura N° 4.13: Diagrama de robustez, caso de uso: Emitir proforma producto tara

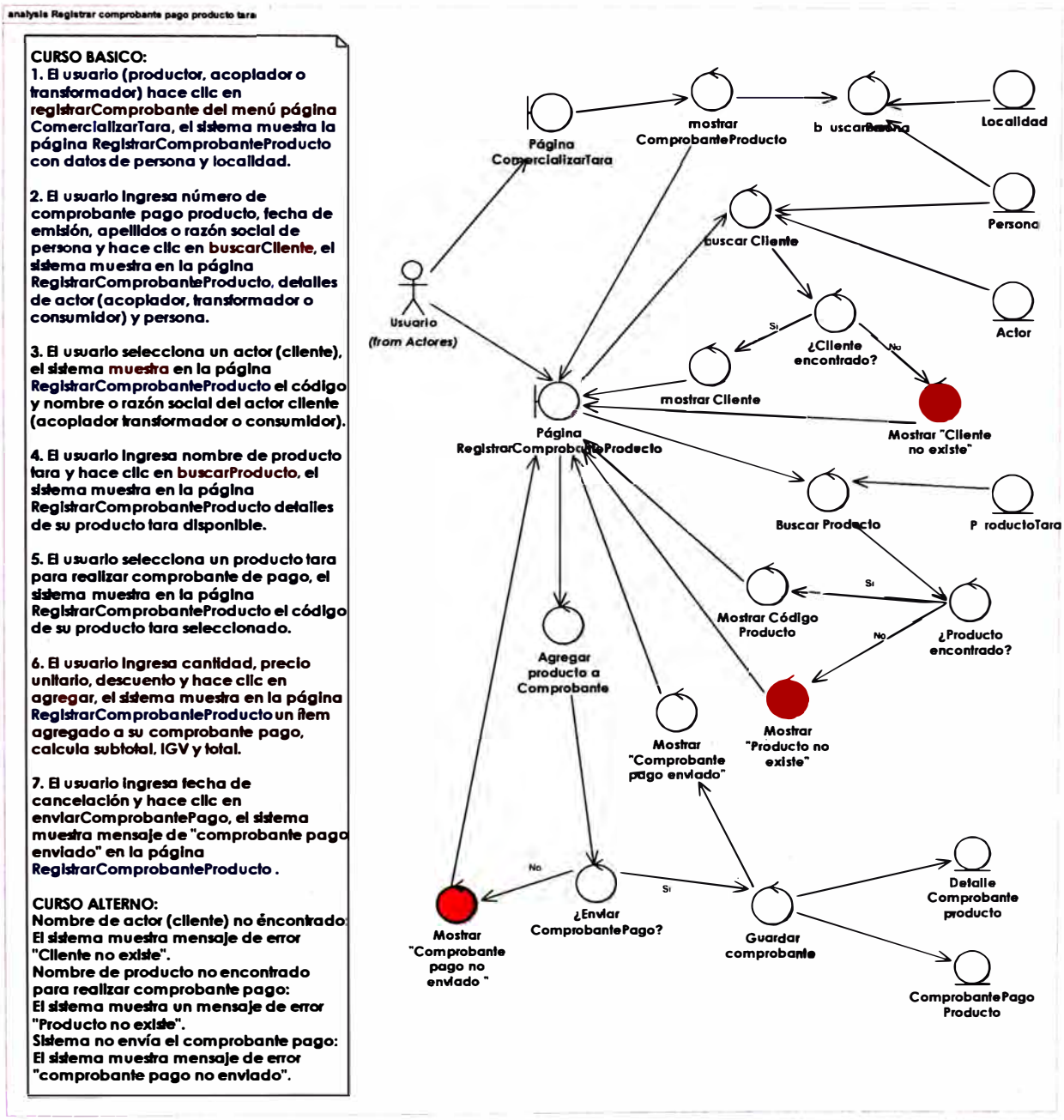

Figura N° 4.14: Diagrama de robustez, caso de uso: Registrar comprobante pago producto tara

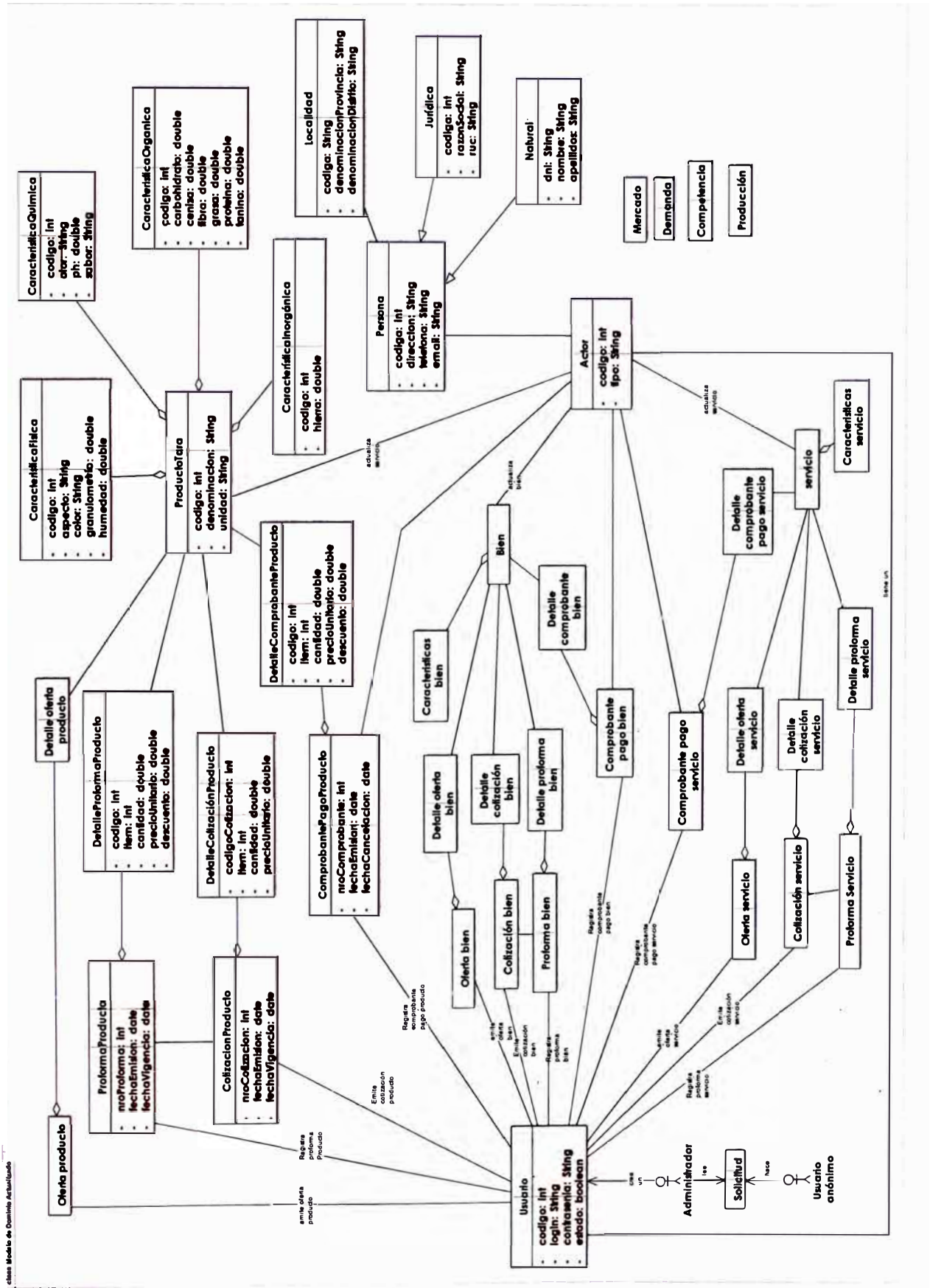

Figura N° 4.15: Modelo de dominio actualizado para dos casos de uso

D. Revisión de Diseño Preliminar (segundo hito)

Durante la revisión no se encontró cambios en los artefactos del modelado del software, para los dos casos de uso, la variación en los modelos que origine los casos de uso adicionales se presentaran en el anexo A.

E. Arquitectura Técnica

Los componentes para implementar los casos de uso, se muestran en la figura 4.16. El componente Dispatcher Servlet de Spring registra todos los controladores (Servlets), y los Beans (Clases Entidad) que serán usados por las interfaces gráficas de JSPs, (Java Server Pages), que a su vez validan las entradas de datos de las interfaces. Los controladores usan las accesadores a datos denominados DAOs (Data access objects), que serán implementados en el componente JDBC (Java Database Connectivity) y los Beans para realizar consultas a la base de datos a través de los DAOs, utilizando el JDBC Template (Clase de acceso a datos) del Spring MVC.

La figura 4.17 muestra los nodos de procesamiento relacionados a los componentes utilizados, donde se observa la arquitectura conformada por el cliente, servidor Web GlassFish y servidor de base de datos postgreSQL.

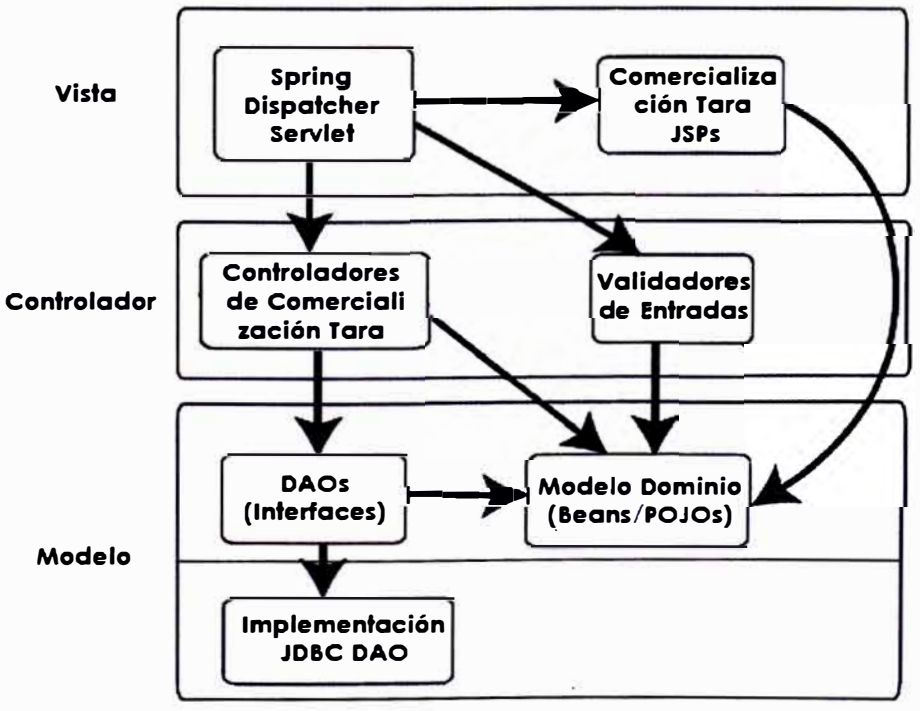

Figura N° 4.16: Diagrama de Componentes - ICONIX

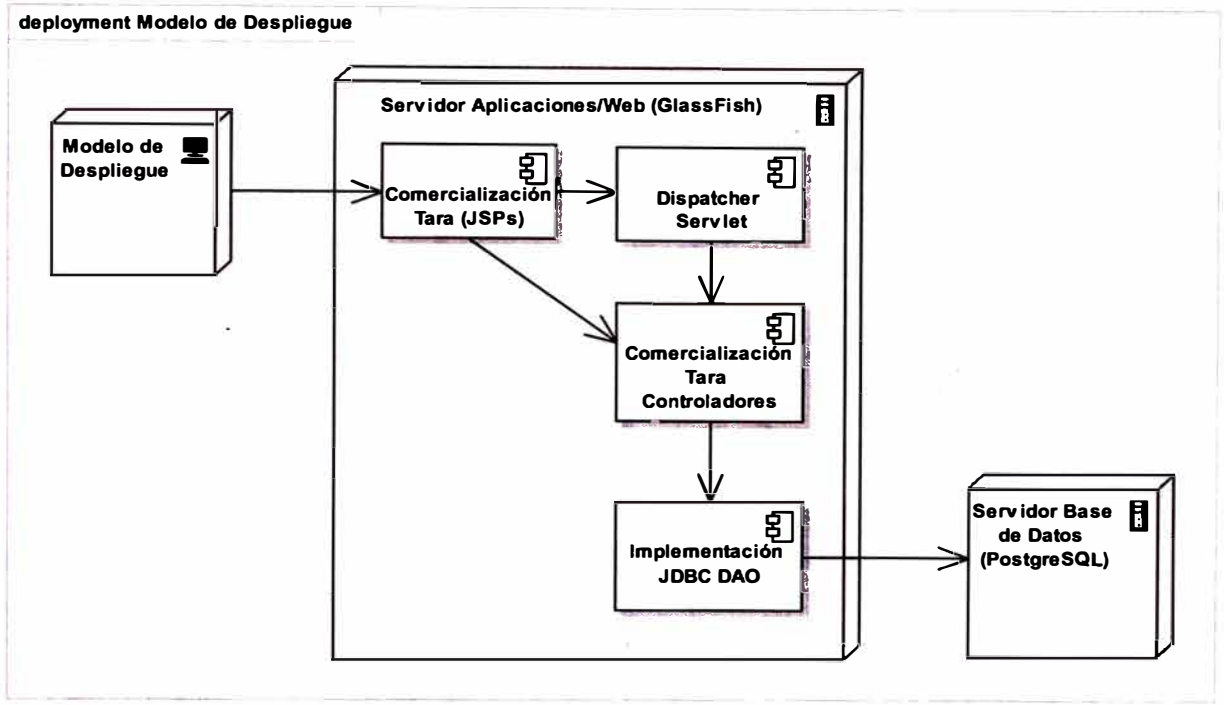

Figura N° 4.17: Diagrama de despliegue - ICONIX

F. Diseño

Luego del análisis de robustez y la definición de la arquitectura técnica, utilizamos el procedimiento descrito en la tabla 3.11 para construir la parte del modelo de dominio actualizado, los diagramas de secuencia, el diagrama de clases orientado a objetos y la lista de controladores identificados para todos los casos de uso.

Los diagramas de secuencia para los casos de uso "registrar comprobante pago producto" y "emitir proforma producto tara", se muestran en el anexo A, figuras A.1 y A.2, el diagrama de clases para dos casos de uso en la figura A.3, el diagrama de clases para todos los casos de uso en la figura A.4 y la base de datos física para el software en la figura A.5.

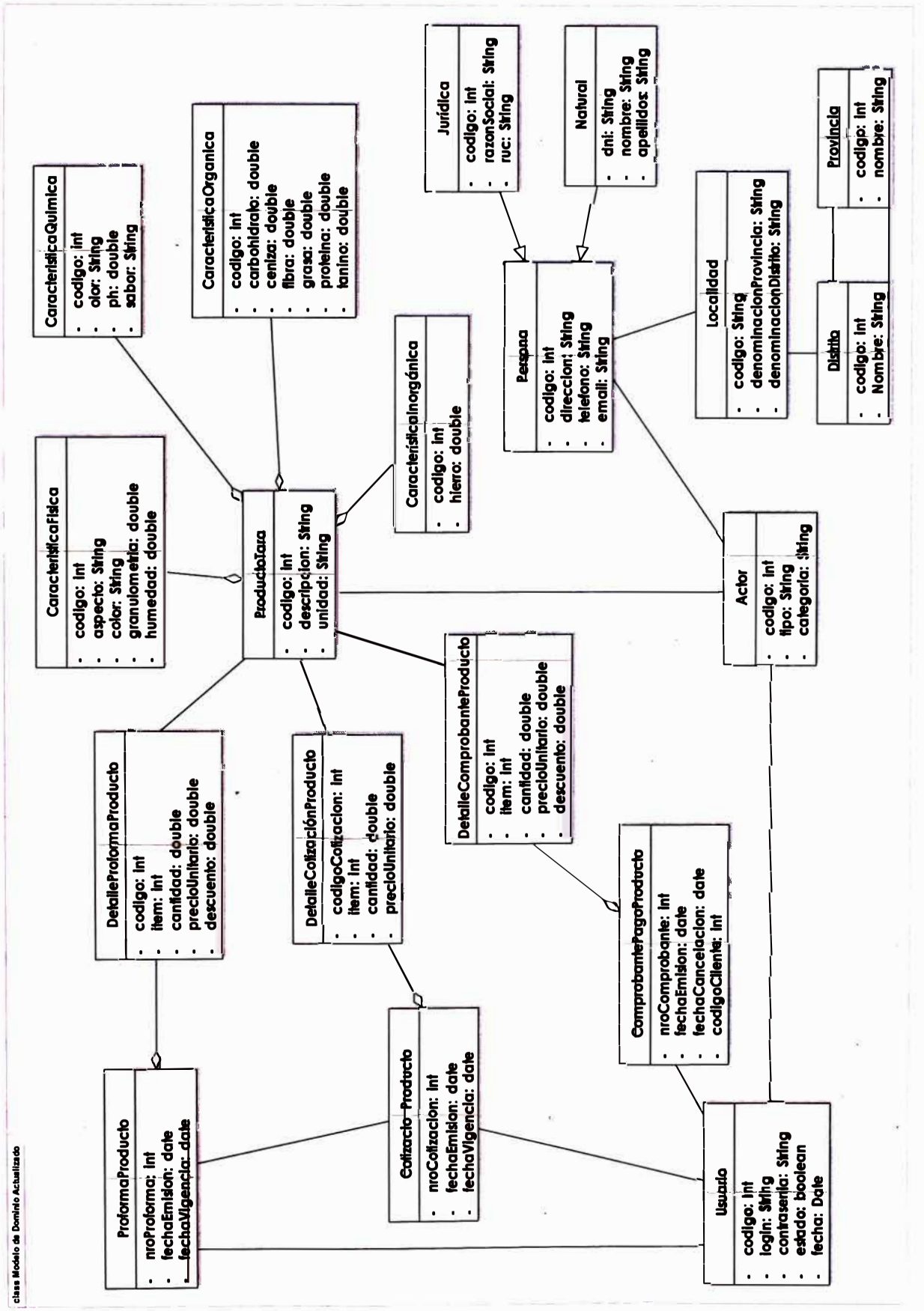

Figura N° 4.18: Modelo de dominio actualizado para dos casos de uso

N° CU	Nombre Clase Control
1	SEmitirProformaProducto.java
2	SMantenerProformaProducto.java
3	SEmitirProformaBien.java
4	SMantenerProformaBien.java
5	SEmitirProformaServicio.java
6	SMantenerProformaServicio.java
7	SRegistrarComprobanteProducto.java
8	SMantenerComprobanteProducto.java
9	SRegistrarComprobanteBien.java
10	SMantenerComprobanteBien.java
11	SRegistrarComprobanteServicio.java
12	SMantenerComprobanteServicio.java
13	SEmitirCotizacionProducto.java
14	SMantenerCotizacionProducto.java
15	SEmitirCotizacionBien.java
16	SMantenerCotizacionBien.java
17	SEmitirCotizacionServicio.java
18	SMantenerCotizacionServicio.java
19	SPublicarOfertaProducto.java
20	SMantenerOfertaProducto.java
21	SPublicarOfertaBien.java
22	SMantenerOfertaBien.java
23	SPublicarOfertaServicio.java
24	SMantenerOfertaServicio.java
25	SSolicitarAcceso.java
26	SLogeo.java
27	SIniciarSesion.java
28	SActualizarCaracterísticaProducto.java
29	SActualizarCaracterísticaBien.java
30	SActualizarCaracterísticaServicio.java
31	SProducto.java
32	SBien.java
33	Sservicio.java

Tabla N° 4.7: Lista de controladores

G. Revisión Critica de Diseño

Se actualizó el diagrama de clases, que se muestra en el anexo A, figura A.4, los diagramas de secuencia para los dos casos de uso no han variado.

H. Implementación

Para implementar utilizaremos las herramientas descritas en la tabla 3.5, siendo: editor de código "NetBeans", lenguaje de programación "Java", servidor web "GlassFish", lenguaje de hipertexto "HTML", validador de campos de texto "JavaScript", pruebas unitarias "Junit", base de datos "PostgreSQL", diseño de interfaces "FrontPage", modelo de implementación "Spring MVC". La base de datos completa se muestra en el anexo A, figura A.5.

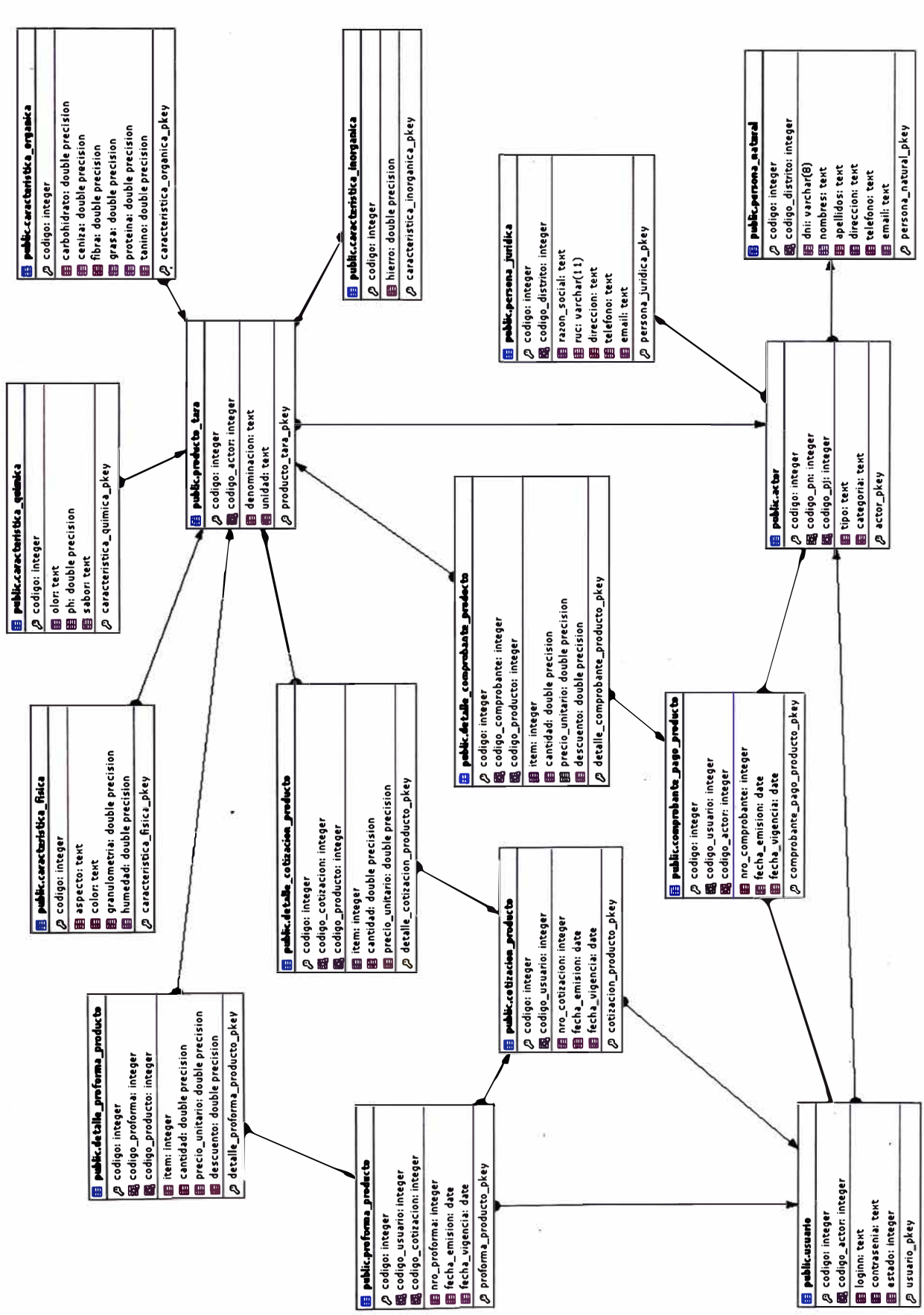

Figura N° 4.19: Base de datos física para dos casos de uso

La figura 4.20 muestra el código fuente para la clase entidad "BDetalleComprobantePago" del caso de uso "registrar comprobante pago producto", declarando atributos, el constructor, los métodos "get" y "set". En java estas clases también se denominan "beans" o "pojos" (clase primitivas).

```
 8   public class BDetalleComprobantePago {
 9       private int codigo;
10       private int item;
11       private double cantidad;
12       private double precio_unitario;
13       private double descuento;
14       private String unidad;
15       private int codigo_proforma;
16       private int codigo_producto;
17       private double subtotal;
18       private double total;
19       private String descrpcion;
20
21       private BSessionBusquedaProducto bSessionBusquedaProducto;
22
23       public BDetalleComprobantePago() {
24           this.bSessionBusquedaProducto = new BSessionBusquedaProducto();
25
26       public int getCodigo() { return codigo;     }
27       public void setCodigo(int codigo) { this.codigo = codigo;     }
28
29       public int getItem() { return item;     }
30       public void setItem(int item) { this.item = item;     }
31
32       public double getCantidad() { return cantidad;     }
33       public void setCantidad(double cantidad) { this.cantidad = cantidad;
34
35       public double getPrecio_unitario() { return precio_unitario;     }
36       public void setPrecio_unitario(double precio_unitario) {
37           this.precio_unitario = precio_unitario;     }
```

Figura N° 4.20: Código fuente de la clase entidad. Detalle comprobante pago

Las interfaces se generaron con FromPage que genera código en HTML, los nombres de las variables para botones, cajas de texto y otros se usan para escribir código de los controladores (servlets), así mismo, se ingresa código java (JSPs) y código scripts para validar las entradas de datos de las interfaces que luego serán utilizados por los controladores y los DAOs.

Para ejecutar una prueba unitaria usamos "JUnit", que prueba los métodos de las clases control, mediante los métodos JUnit "testProcessRequest", "testDoGet", "testDoPost", "testGetServletInfo", "tearDown", "setUp",

"setUpClass", "tearDownClass". La salida se muestra con "AssetTrue" que prueba el valor de verdad y "AssertEquals" que prueba la igualdad de dos expresiones. La figura 4.21 muestra el código fuente de una prueba unitaria.

```
1   package tara.servlet;
2   import javax.servlet.http.HttpServletRequest;
3   import javax.servlet.http.HttpServletResponse;
4   import static org.junit.Assert.*;
5
6   public class SRegistrarComprobanteProductoTest {
7       public SRegistrarComprobanteProductoTest() {     }
8
9       public void testProcessRequest() throws Exception {
10          System.out.println("processRequest");
11          HttpServletRequest request = null;
12          HttpServletResponse response = null;
13          SRegistrarComprobanteProducto instance =
14                  new SRegistrarComprobanteProducto();
15          instance.processRequest(request, response);     }
16
17      public void testDoGet() throws Exception {
18          System.out.println("doGet");
19          HttpServletRequest request = null;
20          HttpServletResponse response = null;
21          SRegistrarComprobanteProducto instance =
22                  new SRegistrarComprobanteProducto();
23          instance.doGet(request, response);     }
24
25      public void testDoPost() throws Exception {
26          System.out.println("doPost");
27          HttpServletRequest request = null;
28          HttpServletResponse response = null;
29          SRegistrarComprobanteProducto instance =
30                  new SRegistrarComprobanteProducto();
31          instance.doPost(request, response);     }
```

Figura N° 4.21: Código fuente de prueba unitaria para el controlador. Detalle comprobante pago

Nº CU	Nombre Clase Control	Nombre de los métodos	Resultado
1	SEmitirProformaProducto.java	processRequest()	Satisfactorio
		calcularSubTotal()	Satisfactorio
		calcularTotal()	Satisfactorio
		doget()	Satisfactorio
		doPost()	Satisfactorio
		getServletInfo()	Satisfactorio
7	SRegistrarComprobanteProducto.java	processRequest()	Satisfactorio
		calcularSubTotal()	Satisfactorio
		calcularTotal()	Satisfactorio
		doget()	Satisfactorio
		doPost()	Satisfactorio
		getServletInfo()	Satisfactorio

Tabla Nº 4.8: Reporte de pruebas unitarias

4.1.2 ARTEFACTOS DEL SOFTWARE APLICANDO EL PROCESO XP

A. Exploración

Aplicando el procedimiento desarrollado en la tabla 3.14, el proceso de XP descrito en el capitulo 2, sección 2.2.3, y la fase de exploración descrita en la sección C.1, capítulo 2. Obtenemos las historias de usuario, arquitectura técnica inicial y el plan de alto nivel.

Nº	Historias de usuario	Descripción
1	Emitir proforma producto tara	El usuario (productor, acopiador y transformador), busca las cotizaciones vigentes de productos tara y emite una proforma para la cotización seleccionada.
2	Mantener proforma producto tara	El usuario busca una proforma de productos tara, para actualizar la fecha de vigencia y detalles de los productos proformado.
3	Emitir proforma bien	El usuario (proveedor de bien), busca las cotizaciones vigentes de bienes y emite una proforma para la cotización seleccionada.
4	Mantener proforma bien	El usuario busca una proforma de bienes, para actualizar la fecha de vigencia y detalles de los bienes proformado.
5	Emitir proforma servicio	El usuario (proveedor de servicio), busca las cotizaciones vigentes de servicios y emite una proforma para la cotización seleccionada.
6	Mantener proforma servicio	El usuario busca una proforma de servicios, para actualizar la fecha de vigencia y detalles de los servicios proformado.
7	Registrar comprobante pago producto tara	El usuario(productor, acopiador y transformador), registra y emite un comprobante de pago para el cliente a quien vendió productos.
8	Mantener comprobante pago producto tara	El usuario busca un comprobante pago de productos tara, para modificar detalles del comprobante pago.
9	Registrar	El usuario (proveedor de bien), registra y emite un

	comprobante pago bien	comprobante de pago para el cliente a quien vendió bienes.
10	Mantener comprobante pago bien	El usuario busca un comprobante pago de bienes, para modificar detalles del comprobante pago.
11	Registrar comprobante pago servicio	El usuario (proveedor de servicio), registra y emite un comprobante de pago para el cliente a quien prestó servicios.
12	Mantener comprobante pago servicio	El usuario busca un comprobante pago de servicios, para modificar detalles del comprobante pago.
13	Emitir cotización producto tara	El usuario (acopiador, transformador y cliente), emite y registra una cotización de productos tara.
14	Mantener cotización producto tara	El usuario busca una cotización de productos tara, para actualizar la fecha de vigencia y detalles de los productos cotizados.
15	Emitir cotización bien	El usuario (productor, acopiador y transformador), emite y registra una cotización de bienes.
16	Mantener cotización bien	El usuario busca una cotización de bienes, para actualizar la fecha de vigencia y detalles de los bienes cotizados.
17	Emitir cotización servicio	El usuario (productor, acopiador y transformador), emite y registra una cotización de servicios.
18	Mantener cotización servicio	El usuario busca una cotización de servicios, para actualizar la fecha de vigencia y detalles de los servicios cotizados.
19	Publicar oferta producto tara	El usuario (productor, acopiador y transformador), publica y registra la oferta de sus productos tara.
20	Mantener oferta producto tara	El usuario busca una oferta de sus productos tara, para actualizar la fecha de vigencia y detalles de los productos.
21	Publicar oferta bien	El usuario (proveedor de bien), publica y registra la oferta de sus bienes.
22	Mantener oferta bien	El usuario busca una oferta de sus bienes, para actualizar la fecha de vigencia y detalles de los bienes.
23	Publicar oferta servicio	El usuario (proveedor de servicio), publica y registra la oferta de sus servicios.
24	Mantener oferta servicio	El usuario busca una oferta de sus servicios, para actualizar la fecha de vigencia y detalles de los servicios.
25	Emitir reporte competencia	El usuario (actor directo), emite el reporte de competencia por provincias de producción y características de los productos tara.
26	Emitir reporte demanda	El usuario (actor directo), emite el reporte de demanda que considera los productos tara demandados.
27	Emitir reporte precios históricos	El usuario (actor directo), emite el reporte por años y provincia, considerando los precios promedio de los productos tara.
28	Solicitar acceso como actor directo o indirecto	El usuario anónimo llena un formato con sus datos personales o institucionales solicitando acceso al sistema de comercialización tara.
29	Mantener cuenta y personalizar actor directo o indirecto	El administrador personaliza una cuenta de acceso como actor directo o indirecto al sistema de comercialización tara.
30	Iniciar sesión	El administrador y los actores directos e indirectos,

		ingresan su login y contraseña para acceder al sistema comercialización tara.
31	Actualizar características producto tara	El usuario (productor, acopiador y transformador), actualiza las características de sus productos tara.
32	Actualizar características bien	El usuario (proveedor de bien), actualiza las características de sus bienes.
33	Actualizar características servicio	El usuario (proveedor de servicios), actualiza las características de sus servicios.
34	Actualizar características producción	El usuario (productor), actualiza las características geográficas, climatologías y de suelos para la producción de la tara.

Tabla N° 4.9: Historias de usuario.

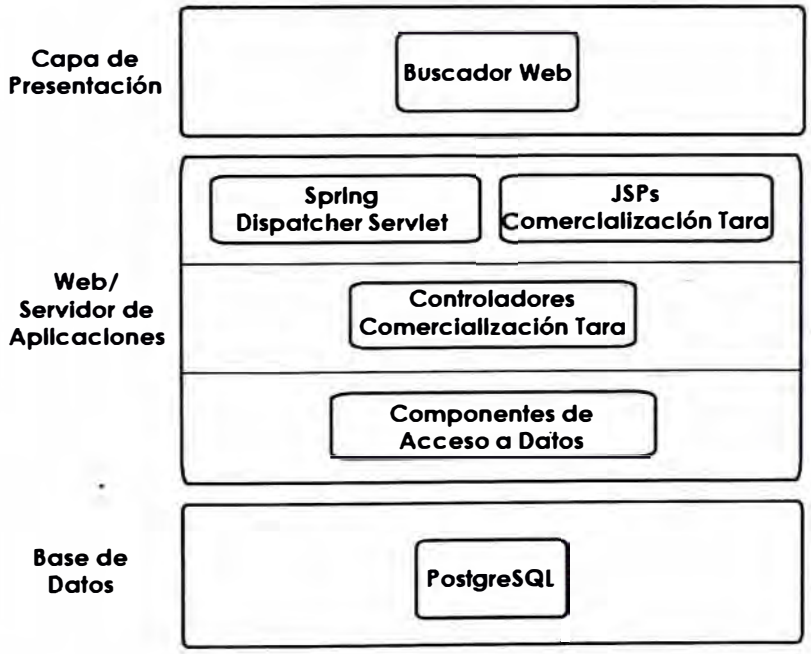

Figura N° 4.22: Arquitectura técnica inicial

N°	Historia de Usuario	Esfuerzo (días)
1	Formular proforma para venta de productos de tara	3
2	Mantener proforma para venta de productos de tara	2
3	Formular proforma para venta de bienes	2
4	Mantener proforma para venta de bienes	3
5	Formular proforma para venta de servicios	2
6	Mantener proforma para venta de servicios	2
7	Registrar comprobante de pago para productos tara	3
8	Mantener comprobante de pago para productos tara	1
9	Registrar comprobante de pago por venta de bienes	2
10	Mantener comprobante de pago por venta de bienes	3
11	Registrar comprobante de pago por venta de servicios	1
12	Mantener comprobante de pago por venta de servicios	3

13	Formular cotización para adquirir productos de tara	1
14	Mantener cotización para adquirir productos de tara	3
15	Formular cotización para adquirir bienes	1
16	Mantener cotización para adquirir bienes	3
17	Formular cotización para adquirir servicios	2
18	Mantener cotización para adquirir servicios	3
19	Publicar oferta para vender productos de tara	1
20	Mantener oferta para vender productos de tara	3
21	Publicar oferta para vender bienes	2
22	Mantener oferta para vender bienes	3
23	Publicar oferta para vender servicios	1
24	Mantener oferta para vender servicios	3
25	Formular reporte de competencia de productores de tara	2
26	Formular reporte de demanda de la tara	1
27	Formular reporte de precios históricos de la tara	1
28	Solicitar acceso como actor directo o indirecto	1
29	Mantener cuenta y personalizar actor directo o indirecto	3
30	Iniciar sesión	1
31	Adicionar características de productos tara	2
32	Adicionar características de bienes	2
33	Adicionar características de servicios	2
34	Adicionar características de producción de la tara	1

Tabla N° 4.10: Plan de alto nivel.

B. Planificación

Usaremos la técnica desarrollada en la tabla 3.15 del proceso XP, en base a la teoría descrita en el capitulo 2, subtítulo 2.2.3 y sección C.2 fase de planificación. Obteniendo los entregables: historias de usuario en detalle para tres historias priorizadas y el plan de versión (una iteración).

Historia de Usuario	
Número: **1**	Usuario: **Productor, Acopiador, Transformador**
Nombre historia: **Formular proforma para venta de productos de tara**	
Prioridad en negocio: **Alta**	Riesgo en desarrollo: **Alta**
Puntos estimados: **3**	Iteración asignada: **1**
Programador responsable: **ROJAS NACCHA, Rossel**	
Descripción: El usuario (productor, acopiador o transformador), elige la opción "emitir proforma" del menú producto, el sistema presenta la interface "emitir proforma producto". El usuario busca y selecciona una cotización vigente para emitir la proforma del producto tara. El usuario busca y selecciona un producto tara, el sistema muestra detalles de su producto, luego ingresa la cantidad, precio unitario y descuento. El usuario adiciona el producto a la proforma, repitiendo los pasos anteriores hasta completar su proforma, finalmente, registra y publica la proforma.	
Observaciones:	

Tabla N° 4.11: Historia usuario. Emitir proforma producto tara

Historia de Usuario	
Número: **7**	Usuario: **Productor, Acopiador, Transformador**
Nombre historia: **Registrar comprobante de pago para productos tara**	
Prioridad en negocio: **Alta**	Riesgo en desarrollo: **Medio**
Puntos estimados: **3**	Iteración asignada: **1**
Programador responsable: **RAMOS GUTIÉRREZ, Miguel Angel**	
Descripción: El usuario (productor, acopiador o transformador), elige la opción "registrar comprobante pago" del menú producto, el sistema presenta la interface "Registrar Comprobante Pago Producto". El usuario busca y selecciona su cliente a quien emitirá el comprobante de pago del producto tara. El usuario busca y selecciona un producto tara vendido, el sistema muestra detalles de su producto, luego ingresa la cantidad, precio unitario y descuento. El usuario adiciona el producto al comprobante pago, repitiendo los pasos anteriores hasta completar su comprobante, finalmente, registra y emite el comprobante pago.	
Observaciones:	

Tabla N° 4.12: Historia usuario. Registrar comprobante pago producto tara

Historia de Usuario	
Número: **30**	Usuario: **todos los usuarios**
Nombre historia: **Iniciar sesión**	
Prioridad en negocio: **Alta**	Riesgo en desarrollo: **Bajo**
Puntos estimados: **2**	Iteración asignada: **1**
Programador responsable: **RAMOS GUTIÉRREZ, Miguel Angel**	
Descripción: El usuario, en la interface "login" ingresan su login y contraseña para iniciar sesión, el sistema le otorga el acceso si los datos son correctos.	
Observaciones:	

Tabla N° 4.13: Historia usuario. Iniciar sesión

N°	Historia de Usuario	Prioridad	Riesgo	Esfuerzo (días)	Iteración
1	Formular proforma para venta de productos de tara	Alta	Alto	3	1
2	Mantener proforma para venta de productos de tara	Baja	Bajo	2	4
3	Formular proforma para venta de bienes	Media	Alta	2	2
4	Mantener proforma para venta de bienes	Baja	Bajo	3	4
5	Formular proforma para venta de servicios	Media	Alto	2	3
6	Mantener proforma para venta de servicios	Baja	Bajo	2	4
7	Registrar comprobante de pago para productos tara	Alta	Medio	3	1
8	Mantener comprobante de pago para productos tara	Baja	Bajo	1	4
9	Registrar comprobante de pago por venta de bienes	Media	Alto	2	2
10	Mantener comprobante de pago	Baja	Bajo	3	4

	por venta de bienes				
11	Registrar comprobante de pago por venta de servicios	Media	Alto	1	2
12	Mantener comprobante de pago por venta de servicios	Baja	Bajo	3	3
13	Formular cotización para adquirir productos de tara	Baja	Alto	1	4
14	Mantener cotización para adquirir productos de tara	Media	Bajo	3	2
15	Formular cotización para adquirir bienes	Baja	Alto	1	4
16	Mantener cotización para adquirir bienes	Media	Bajo	3	2
17	Formular cotización para adquirir servicios	Baja	Alta	2	4
18	Mantener cotización para adquirir servicios	Baja	Bajo	3	3
19	Publicar oferta para vender productos de tara	Media	Alto	1	2
20	Mantener oferta para vender productos de tara	Baja	Bajo	3	4
21	Publicar oferta para vender bienes	Media	Alto	2	2
22	Mantener oferta para vender bienes	Baja	Bajo	3	4
23	Publicar oferta para vender servicios	Media	Alto	1	3
24	Mantener oferta para vender servicios	Baja	Bajo	3	4
25	Formular reporte de competencia de productores de tara	Media	Medio	2	5
26	Formular reporte de demanda de la tara	Baja	Medio	1	5
27	Formular reporte de precios históricos de la tara	Media	Medio	1	5
28	Solicitar acceso como actor directo o indirecto	Baja	Medio	1	3
29	Mantener cuenta y personalizar actor directo o indirecto	Media	Medio	3	5
30	Iniciar sesión	Alta	Bajo	1	1
31	Adicionar características de productos tara	Baja	Medio	2	5
32	Adicionar características de bienes	Baja	Medio	2	5
33	Adicionar características de servicios	Baja	Medio	2	5
34	Adicionar características de producción de la tara	Baja	Bajo	1	5

Tabla N° 4.14: Plan de versión

C. Iteración

La fase de iteración desarrollado en la sección C.3, subtítulo 2.2.3 del capitulo II, y la técnica descrita en la tabla 3.16 permitirá obtener los siguientes entregables: arquitectura técnica, tareas de ingeniería, plan de iteración, casos de prueba de aceptación, GUI, tarjetas CRC, base de datos física, código

fuente para clases entidad, pruebas unitarias, código fuente para tarea de ingeniería, reporte de pruebas unitarias, reporte de pruebas de integración y de aceptación.

La figura 4.23, muestra los componentes para implementar una historia de usuario, el componente Dispatcher Servlet de Spring se registran todos los controladores, que serán usados por el Web flow, el componente Web flow utiliza los JSPs (Java Server Pages) y los Beans (clases entidad) para mostrar la interfaz gráfica de usuario, los controladores usan las interfaces de los DAOs (Data access objects) y los Beans para realizar accesos a la base de datos, las interfaces de los DAOs se implementarán en el componente JDBC(Java Database Connectivity) DAO, utilizando el JDBC Template (clase de acceso a datos) de Spring.

La figura 4.24 del diagrama de despliegue organiza los nodos de procesamiento con sus componentes utilizados, observamos la arquitectura conformada por el cliente, servidor Web GlassFish y servidor de base de datos postgreSQL.

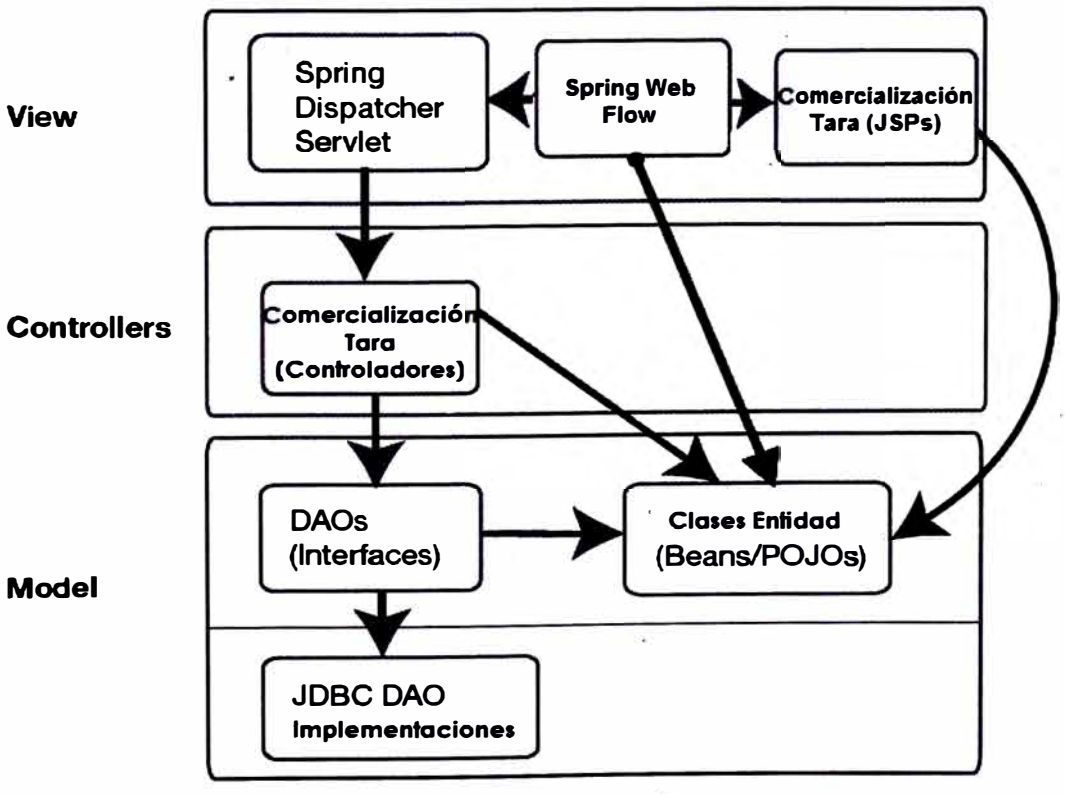

Figura N° 4.23: Arquitectura técnica. Diagrama de componentes

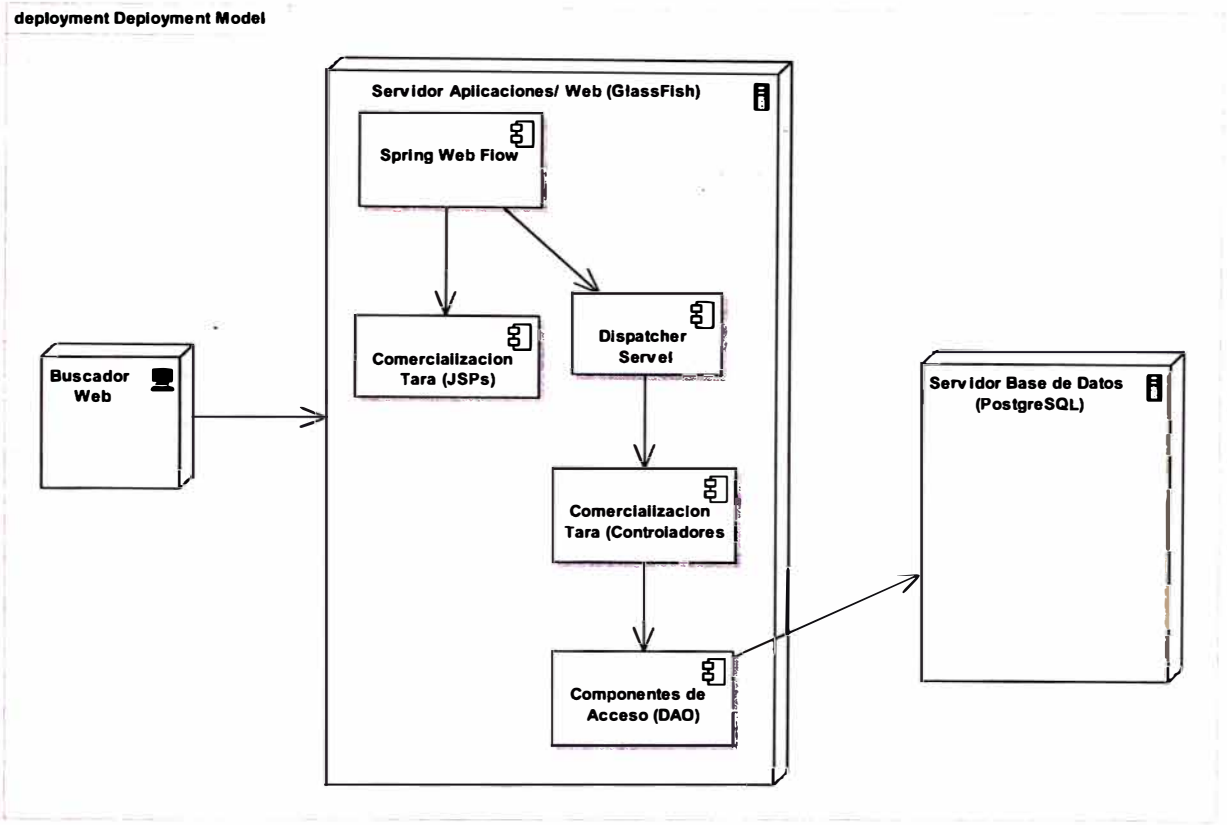

Figura N° 4.24: Arquitectura técnica. Diagrama de despliegue

Tarea de Ingeniería	
Número tarea de ingeniería: **1**	Número historia de usuario: **30**
Nombre tarea: **Autenticar Usuario**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **01/09/2009**	Fecha fin: **03/09/2009**
Programador responsable: **ROJAS NACCHA, Rossel**	
Descripción: El Usuario ingresa su login y contraseña, el sistema busca y compara si existe el login y contraseña en la base de datos de usuarios. Si los datos son correctos, el sistema muestra la pagina principal para comercializar producto tara, con las historias de usuario permitidas de acuerdo al actor directo o indirecto. Si los datos son incorrectos o no existen, el sistema muestra un mensaje de error de login o contraseña.	

Tabla N° 4.15: Tarea de ingeniería. Autenticar usuario

Tarea de Ingeniería	
Número tarea de ingeniería: **1**	Número historia de usuario: **1**
Nombre tarea: **Buscar Cotización Producto**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **01/09/2009**	Fecha fin: **03/09/2009**
Programador responsable: **RAMOS GUTIERREZ, Miguel Angel**	
Descripción: El Usuario hace clic en el botón "buscar cotización" de la página "Emitir Proforma Producto", el sistema muestra la pagina "Buscar Cotización", el usuario ingresa nombre de su producto o deja en blanco y hace clic en el botón "buscar", el sistema muestra las cotizaciones vigentes.	

El usuario selecciona una cotización para proformar, el sistema muestra el detalle de los productos cotizados.

Tabla N° 4.16: Tarea de ingeniería. Buscar cotización producto

Tarea de Ingeniería	
Número tarea de ingeniería: **2**	Número historia de usuario: **1**
Nombre tarea: **Buscar Producto Tara**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **03/09/2009**	Fecha fin: **05/09/2009**
Programador responsable: **ROJAS NACCHA, Rossel**	
Descripción: El Usuario hace clic en el botón "buscar producto" de la página "Emitir Proforma Producto", el sistema muestra la pagina "Buscar Producto", el usuario ingresa nombre de su producto o deja en blanco y hace clic en el botón "buscar", el sistema muestra sus productos. El usuario selecciona un producto para proformar, el sistema muestra detalles del producto tara para proformar.	

Tabla N° 4.17: Tarea de ingeniería. Buscar producto tara

Tarea de Ingeniería	
Número tarea de ingeniería: **3**	Número historia de usuario: **1**
Nombre tarea: **Agregar detalle proforma producto**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **03/09/2009**	Fecha fin: **05/09/2009**
Programador responsable: **RAMOS GUTIERREZ, Miguel Angel**	
Descripción: El usuario en la pagina "Emitir Proforma Producto", ingresa cantidad, precio unitario, descuento del producto y hace clic en el botón "Agregar Producto", el sistema agrega un producto con sus detalles a la proforma y calcula subtotal, IGV y total.	

Tabla N° 4.18: Tarea de ingeniería. Agregar detalle proforma producto

Tarea de Ingeniería	
Número tarea de ingeniería: **4**	Número historia de usuario: **1**
Nombre tarea: **Guardar Proforma Producto**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **05/09/2009**	Fecha fin: **07/09/2009**
Programador responsable: **ROJAS NACCHA,Rossel**	
Descripción: El Usuario en la pagina "Emitir Proforma Producto", ingresa fecha de vigencia, fecha de emisión y numero de la proforma y hace clic en el botón "Enviar Proforma" y el sistema registra y publica la proforma y muestra un mensaje de "Registrado con éxito".	

Tabla N° 4.19: Tarea de ingeniería. Guardar Proforma Producto

Tarea de Ingeniería	
Número tarea de ingeniería: **1**	Número historia de usuario: **7**
Nombre tarea: **Buscar Cliente**	
Tipo de tarea : **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **05/09/2009**	Fecha fin: **07/09/2009**
Programador responsable: **RAMOS GUTIERREZ, Miguel Angel**	
Descripción:	

El Usuario hace clic en el botón "Buscar Cliente" de la página "Registrar Comprobante de Pago de Producto", el sistema muestra la pagina "Buscar Cliente", el usuario ingresa nombre del cliente o deja en blanco y hace clic en el botón "buscar", el sistema muestra los clientes. El usuario selecciona un cliente a quien emite un comprobante, el sistema muestra datos del cliente.

Tabla N° 4.20: Tarea de ingeniería. Registrar comprobante pago producto tara

Tarea de Ingeniería	
Número tarea de ingeniería: **2**	Número historia de usuario: **7**
Nombre tarea: **Agregar detalle comprobante producto**	
Tipo de tarea: **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **07/09/2009**	Fecha fin: **09/09/2009**
Programador responsable: **ROJAS NACCHA, Rossel**	
Descripción: El usuario en la pagina "Registrar Comprobante de Pago de Producto", ingresa cantidad, precio unitario, descuento del producto y hace clic en el botón "Agregar Producto", el sistema agrega un producto y sus detalles al comprobante y calcula subtotal, IGV y total.	

Tabla N° 4.21: Tarea de ingeniería. Agregar detalle comprobante producto

Tarea de Ingeniería	
Número tarea de ingeniería: **3**	Número historia de usuario: **7**
Nombre tarea: **Guardar Comprobante producto**	
Tipo de tarea: **Desarrollo**	Puntos estimados: **3**
Fecha inicio: **07/09/2009**	Fecha fin: **09/09/2009**
Programador responsable: **RAMOS GUTIERREZ, Miguel Angel**	
Descripción: El Usuario en la pagina "Registrar Comprobante de Pago de Producto", ingresa fecha de cancelación, fecha de emisión y numero de comprobante y hace clic en el botón "Enviar Comprobante", el sistema registra y emite el comprobante y muestra un mensaje de "Registrado con éxito".	

Tabla N° 4.22: Tarea de ingeniería. Guardar comprobante producto

N° HU	N° TI	Fecha Inicio	Fecha Fin	Programador
30	1	01/09/2009	03/09/2009	ROJAS NACCHA,Rossel
1	1	01/09/2009	03/09/2009	RAMOS GUTIERREZ, Miguel Angel
1	2	03/09/2009	05/09/2009	ROJAS NACCHA,Rossel
1	3	03/09/2009	05/09/2009	RAMOS GUTIERREZ, Miguel Angel
1	4	05/09/2009	07/09/2009	ROJAS NACCHA,Rossel
7	1	05/09/2009	07/09/2009	RAMOS GUTIERREZ, Miguel Angel
7	2	07/09/2009	09/09/2009	ROJAS NACCHA,Rossel
7	3	07/09/2009	09/09/2009	RAMOS GUTIERREZ, Miguel Angel

Tabla N° 4.23: Plan de iteración (Primera)

N° HU	Requisito	N° CP	Caso de Prueba (CP)
1	1. El sistema debe ser capaz de mostrar los datos del actor proveedor de producto.	1	Verificar el apellido o razón social, actor, localidad, dirección, teléfono y RUC o DNI, del actor proveedor de producto tara.
		2	Verificar que la cotización

	2. El sistema debe ser capaz de buscar una o varias cotizaciones vigentes. 3. El sistema debe buscar uno o varios productos tara que pertenecen al actor proveedor. 4. El sistema debe calcular los subtotales y total de la proforma. 5. El sistema debe ser capaz de imprimir la proforma.		seleccionada, pertenece a un actor (transformador, acopiador y cliente) y tiene fecha de vigencia, descripción, unidad, cantidad correcto de los productos mostrados.
		3	Verificar que el producto tara seleccionado, pertenece a un actor proveedor (productor, transformador y acopiador) y tiene código, descripción y características correctas.
		4	Comprobar que la cantidad, el precio unitario, descuento, subtotal y total son correctos.
		5	Verificar que los datos de la proforma se grabaron correctamente imprimiendo la proforma.
7	1. El sistema debe ser capaz de mostrar los datos del actor proveedor de producto. 2. El sistema debe ser capaz de buscar uno o varios clientes. 3. El sistema debe buscar uno o varios productos tara que pertenecen al actor proveedor. 4. El sistema debe calcular los subtotales y total del comprobante. 5. El sistema debe ser capaz de imprimir el comprobante.	1	Verificar el apellido o razón social, actor, localidad, dirección, teléfono y RUC o DNI, del actor proveedor de producto tara.
		2	Verificar el código, apellido o razón social, actor, dirección, RUC o DNI del actor cliente seleccionado (transformador, acopiador y cliente), sean correctos.
		3	Verificar que el producto tara seleccionado, pertenece a un actor proveedor (productor, transformador y acopiador) y tiene código, descripción y características correctas.
		4	Comprobar que la cantidad, el precio unitario, descuento, subtotal y total son correctos.
		5	Verificar que los datos de la proforma se grabaron correctamente imprimiendo el comprobante.
30	1. El sistema debe ser capaz de iniciar una sesión para un usuario.	1	Acceder al sistema ingresando el login y la contraseña.

Tabla N° 4.24: Casos de prueba de aceptación

Artefacto GUI.- Las interfaces para ICONIX y XP son similares, usamos las mismas técnicas de diseño de interfaces con la participación del cliente, las interfaces para las tres historias de usuario se muestran en las figuras 4.11 y 4.12.

Clase: Usuario	
Responsabilidades: Verificar login, cotraseña y estado. Mostrar tipo de actor, apellido, razón social, dirección, DNI o RUC de persona.	**Colaboradores:** Actor Persona

Tabla N° 4.25: Tarjeta CRC. Iniciar sesión

Clase: Actor	
Responsabilidades: Buscar tipo de actor. Enviar tipo de actor, apellido, razón social, dirección, DNI o RUC de persona a usuario.	**Colaboradores:** Usuario Persona

Tabla N° 4.26: Tarjeta CRC. Iniciar sesión

Clase: Persona	
Responsabilidades: Buscar apellido, razón social, dirección, DNI o RUC de persona. Enviar apellido, razón social, dirección, DNI o RUC de persona a actor.	**Colaboradores:** Actor

Tabla N° 4.27: Tarjeta CRC. Iniciar sesión

Clase: ProductoTara	
Responsabilidades: Buscar producto tara por actor directo. Mostrar denominación y características del producto tara.	**Colaboradores:** Actor CaracteristicaFisica CaracteristicaQuimica CaracteristicaOrganica Caracteristicalnorganica

Tabla N° 4.28: Tarjeta CRC. Emitir proforma producto tara

Clase: CaracteristicaFisica	
Responsabilidades: Buscar característica física. Enviar característica física a producto tara.	**Colaboradores:** ProductoTara

Tabla N° 4.29: Tarjeta CRC. Emitir proforma producto tara

Clase: CaracteristicaQuimica	
Responsabilidades: Buscar característica química. Enviar característica química a producto tara.	**Colaboradores:** ProductoTara

Tabla N° 4.30: Tarjeta CRC Emitir proforma producto tara

Clase: CaracteristicaOrganica	
Responsabilidades: Buscar característica orgánica. Enviar característica orgánica a producto tara.	**Colaboradores:** ProductoTara

Tabla N° 4.31: Tarjeta CRC. Emitir proforma producto tara

Clase: Caracteristicalnorganica	
Responsabilidades: Buscar característica inorgánica. Enviar característica inorgánica a producto tara.	**Colaboradores:** ProductoTara

Tabla N° 4.32: Tarjeta CRC. Emitir proforma producto tara

Clase: DetalleProformaProducto	
Responsabilidades: Obtener código de producto tara. Obtener número de proforma. Guardar código de producto, ítem, cantidad, precio unitario, descuento, número de proforma.	**Colaboradores:** ProductoTara. ProformaProducto.

Tabla N° 4.33: Tarjeta CRC. Emitir proforma producto tara

Clase: ProformaProducto	
Responsabilidades: Enviar número de proforma a detalle proforma producto. Obtener código de cotización producto. Guardar número de proforma, fecha de emisión y fecha de vigencia.	**Colaboradores:** DetalleProformaProducto CotizacionProducto

Tabla N° 4.34: Tarjeta CRC. Emitir proforma producto tara

Clase: CotizacionProducto	
Responsabilidades: Buscar número de cotización, fecha de vigencia. Buscar código y tipo de actor. Mostrar detalle de cotización producto.	**Colaboradores:** Usuario Actor

Tabla N° 4.35: Tarjeta CRC. Emitir proforma producto tara

Clase: DetalleCotizacionProducto	
Responsabilidades: Buscar código de producto, cantidad y precio unitario. Buscar código de producto	**Colaboradores:** ProductoTara CotizacionProducto

Tabla N° 4.36: Tarjeta CRC. Emitir proforma producto tara

Clase: DetalleComprobanteProducto	
Responsabilidades: Obtener código de producto tara. Obtener número de comprobante. Guardar código de producto, ítem, cantidad, precio unitario, descuento, número de comprobante.	**Colaboradores:** ProductoTara. ComprobanteProducto.

Tabla N° 4.37: Tarjeta CRC. Registrar comprobante pago producto tara

Clase: ComprobanteProducto	
Responsabilidades: Enviar número de comprobante a detalle comprobante producto. Guardar número de comprobante, fecha de emisión y fecha de cancelación.	**Colaboradores:** DetalleComprobanteProducto

Tabla N° 4.38: Tarjeta CRC. Registrar comprobante pago producto tara

La figura 4.25 muestra el código fuente para las clases entidad, de la historia de usuario "emitir proforma producto", para la clase "ProformaProducto", declarando atributos, el constructor y los métodos "get" y "set". En java estas clases también se denominan "beans" o "pojos" (clase primitivas).

```
11    public class ProformaProducto  implements java.io.Serializable {
12
13
14          private int codigo;
15          private CotizacionProducto cotizacionProducto;
16          private Usuario usuario;
17          private int nroProforma;
18          private Date fechaEmision;
19          private Date fechaVigencia;
20          private List detalleProformaProductos;
21
22          public ProformaProducto() {
23              this.usuario = new Usuario();
24              this.cotizacionProducto = new CotizacionProducto();
25              this.detalleProformaProductos = new LinkedList();
26          }
27
28          public int getCodigo() {
29              return codigo;
30          }
31
32          public void setCodigo(int codigo) {
33              this.codigo = codigo;
34          }
35
36          public CotizacionProducto getCotizacionProducto() {
37              return cotizacionProducto;
38          }
39
40          public void setCotizacionProducto(CotizacionProducto cotizaci
```

Figura N° 4.25: Código fuente de la clase entidad. Proforma producto

La prueba unitaria se realiza con el framework "JUnit", para probar los métodos de las clases control, usando los métodos de JUnit "testProcessRequest", "testDoGet", "testDoPost", "testGetServletInfo", "tearDown", "setUp", "setUpClass", "tearDownClass". La salida de la prueba se verifica con "AssetTrue" que prueba el valor de verdad y "AssertEquals" que prueba la igualdad de dos expresiones.

La figura 4.26, muestra la implementación usando el framework Spring Web Flow, para la configuración del flujo "proforma-producto-flow", con los

siguientes estados: estado de inicio <start-state>, estado de vista <view-state> diseccionando a la pagina de "emitir proforma producto", el estado de acción <action-state> para controlar eventos, direccionamiento hacia clases control registrados en "Dispatcher Servlet" y, estado final del flujo <end-state>.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <flow xmlns="http://www.springframework.org/schema/webflow"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xsi:schemaLocation="http://www.springframework.org/schema/webflow
5                            http://www.springframework.org/schema/webflow/spring-webflow-
6  <var name="producto" class="org.tara.bean.ProductoTara" scope="flow"/>
7  <var name="cotizacionproducto" class="org.tara.bean.CotizacionProducto" scope="flow"/>
8  <var name="proformarproducto" class="org.tara.bean.ProformaProducto" scope="flow"/>
9
10     <!--recupera el usuario-->
11     <input-mapper>
12               <mapping source="usuario" target="flowScope.usuario" />
13        </input-mapper>
14
15        <start-state idref="refProformarProducto"/>
16
17     <view-state id="refProformarProducto" view="emitir-proforma-producto">
18           <transition on="sub_ctz" to="buscar-cotizacion"/><!--boton para buscar coti
19           <transition on="buscar-producto" to="searchProducto"/><!--boton para mostra
20           <transition on="add_producto" to="addProducto"/><!--boton para agregar deta
21           <transition on="emitir_proforma" to="emitirProforma"/><!--boton para guarda
22           <transition on="cancelar" to="terminar"/><!--boton para cancelar proforma--
23     </view-state>
```

Figura N° 4.26: Código fuente para el historia de usuario. Emitir proforma producto

N° HU	N° TI	Nombre Clase Control	Resultado
30	1	SeguridadAction (autentica usuario)	Satisfactorio
1	1	ActionBuscarCotizacionProducto	Satisfactorio
	1	ActionSelecCotizacionProducto	Satisfactorio
	2	ActionBuscarProducto	Satisfactorio
	2	ActionSelecProducto	Satisfactorio
	3	ActionAddProductoProforma (agrega detalle)	Satisfactorio
	4	ActionGuardarProformarProducto (almacena en BD)	Satisfactorio
7	1	ActionBuscarCliente	Satisfactorio
	1	ActionSelecCliente	Satisfactorio
	2	ActionAddComprobantePago (agrega detalle)	Satisfactorio
	3	ActionGuardarComprobantePago (graba la BD)	Satisfactorio

Tabla N° 4.39: Reporte de pruebas unitarias. Primera iteración

N° HU	N° TI	Nombre Tarea de ingeniería	Resultado
30	1	Autenticar usuario	Satisfactorio
1	1	Buscar cotización producto	Satisfactorio
	2	Buscar producto tara	Satisfactorio
	3	Agregar detalle proforma producto	Satisfactorio
	4	Guardar proforma producto	Satisfactorio
7	1	Buscar cliente	Satisfactorio
	2	Agregar detalle comprobante producto	Satisfactorio
	3	Guardar comprobante producto	Satisfactorio

Tabla N° 4.40: Reporte de prueba integración. Primera iteración

N° CASO PRUEBA	1
Propósito	Verificar datos del proveedor del producto
ACTIVIDAD	
Inicialización	Seleccionar proveedor
Descripción de datos de entrada	El usuario hace clic en la opción "Proformar" del menú, ingresa a la página "Emitir proforma producto".
RESULTADOS	
Esperados	Rojas Tello, Carlos. Productor. Ayacucho-Huamanga. Sucre 145. 323232. 1234567
Reales	Rojas Tello, Carlos. Productor. Ayacucho-Huamanga. Sucre 145. 323232. 1234567

Tabla N° 4.41: Reporte de prueba de aceptación. Emitir Proforma Producto

N° CASO PRUEBA	2
Propósito	Verificar datos de cotización seleccionada.
ACTIVIDAD	
Inicialización	Seleccionar cotización.
Descripción de los datos de entrada	El usuario en la página "buscar cotización" ingresa el nombre del producto o deja en blanco y selecciona una cotización
RESULTADOS	
Esperados	43058336, Rossel Rojas Naccha, Productor. 2009-09-27, goma, gr, 50.0
Reales	43058336, Rossel Rojas Naccha, Productor. 2009-09-27, goma, gr, 50.0323232. 1234567

Tabla N° 4.42: Reporte de prueba de aceptación. Emitir Proforma Producto

N° CASO PRUEBA	3
Propósito	Verificar datos de producto seleccionado.
ACTIVIDAD	
Inicialización	Seleccionar producto
Descripción de los datos de entrada	El usuario en la página "buscar producto" ingresa el nombre del producto o deja en blanco y selecciona un producto
RESULTADOS	
Esperados	Carlos Rojas Tello. goma de tara, fibra=0.85, proteína=41.2
Reales	Carlos Rojas Tello. goma de tara, fibra=0.85, proteína=41.2

Tabla N° 4.43: Reporte de prueba de aceptación. Emitir Proforma Producto

Nº CASO PRUEBA	4
Propósito	Verificar cálculos del detalle de la proforma producto.
ACTIVIDAD	
Inicialización	Adicionar productos al detalle de la proforma
Descripción de los datos de entrada	El usuario en la página "Emitir proforma producto" ingresa cantidad, precio unitario, descuento, subtotal y total para cada producto.
RESULTADOS	
Esperados	Goma de tara, kg. 10.0, 5.0, 10.0, 45.0, 45.0
Reales	Goma de tara, kg. 10.0, 5.0, 10.0, 45.0, 45.0

Tabla N° 4.44: Reporte de prueba de aceptación. Emitir Proforma Producto

Nº CASO PRUEBA	5
Propósito	Verificar que la proforma producto se imprime correctamente.
ACTIVIDAD	
Inicialización	Guardar la proforma producto
Descripción de los datos de entrada	Número de proforma
RESULTADOS	
Esperados	30, 24/09/2009 Goma de tara, kg. 10.0, 5.0, 10.0, 45.0, 45.0
Reales	30, 24/09/2009 Goma de tara, kg. 10.0, 5.0, 10.0, 45.0, 45.0

Tabla N° 4.45: Reporte de prueba de aceptación. Emitir Proforma Producto

Nº CASO PRUEBA	1
Propósito	Acceder al sistema comercialización tara
ACTIVIDAD	
Inicialización	Ejecutar el sistema comercialización tara, e ingresar a la página de logueo.
Descripción de los datos de entrada	Login, contraseña
RESULTADOS	
Esperados	Carlos Rojas Tello, Producto
Reales	Accede al sistema comercialización de tara.

Tabla N° 4.46: Reporte de prueba de aceptación. Emitir Proforma Producto

4.1.3 PROCEDIMIENTO PARA INSTALAR LAS APLICACIONES WEB

SmartFTP o cliente FTP (protocolo de transferencia de archivos) se ejecuta en nuestra computadora, usa el protocolo FTP para conectarse a un servidor FTP y transferir archivos de nuestra computadora a un servidor en Internet. Para transferir los archivos de la aplicación proceder como sigue:

a1. Para acceder al servidor FTP remoto, en la barra de herramientas ingresar la dirección "www.comercializacióntara.org.pe" en el campo dominio FTP, luego ingresar nombre de usuario, contraseña, número del puerto para transferir los archivos y presionar conectar.

a2. El cliente FTP muestra una interfase con archivos existentes en el servidor y nuestro computador.

a3. Para transferir el software para comercializar tara al servidor Web, arrastramos la carpeta "ComercializarTaraICONIX" o "ComercializarTaraXP" desde nuestro disco hacia el servidor FTP, al finalizar la transferencia de los archivos, el sistema muestra el mensaje de "transferencia realizada con éxito".

a4. Para que se ejecute el archivo "index.htm" de la aplicación ICONIX o XP, se debe transferir estos archivos fuera de las carpetas "ComercializarTaraICONIX" o "ComercializarTaraXP", para que reconozca el dominio www.comercializacióntara.org.pe. Siguiendo el procedimiento a3.

a5. Al adquirir el hosting, debemos especificar la base de datos PostgreSQL Ver. 8.4, para transferir los archivos que contiene el código script SQL del software para comercializar tara de ICONIX o XP al servidor de base de datos, realizarlo de forma similar al procedimiento a3.

4.1.4 EVALUACION DE LA CALIDAD DEL PRODUCTO SOFTWARE

Las tablas 4.47 y 4.48 se han formulado, considerando el aspecto teórico del capítulo II, subtítulo 2.2.4.4, el capitulo III, subtítulo 3.5.2 y el aspecto práctico para los requerimientos de calidad del usuario. El modelo de calidad seleccionado de acuerdo a la norma NTP ISO/IEC 9126, se determinó los pesos de características y sub características para las necesidades del usuario, en consideración a que la comercialización de la tara en la Región Ayacucho, no cuenta con apoyo financiero, tiene recursos humanos limitados, no tiene infraestructura tecnológica específica y es software pequeño - mediano.

CARACTERISTICA	PESO CARACTERISTICA
Funcionalidad	20%
Fiabilidad	20%
Usabilidad	20%

Eficiencia	10%
Facilidad de Mantenimiento	10%
Portabilidad	20%

Tabla N° 4.47: Pesos de características de las necesidades de usuarios

CALIDAD EXTERNA E INTERNA		
CARACTERISTICA	**SUB CARACTERISTICA**	**PESO SUB CARACTERISTICA**
Funcionalidad	Aplicabilidad	40.00%
	Precisión	60.00%
Fiabilidad	Madurez	45.00%
	Tolerancia a fallas	55.00%
Usabilidad	Entendibilidad	30.00%
	Facilidad de aprendizaje	35.00%
	Operabilidad	35.00%
Eficiencia	Comportamiento en el tiempo	60.00%
	Utilización de recursos	40.00%
Facilidad de Mantenimiento	Cambiabilidad	100.00%
Portabilidad	Adaptabilidad	60.00%
	Instalabilidad	40.00%

Tabla N° 4.48: Pesos de sub características de las necesidades de usuarios

La tabla 4.49 muestra las métricas relacionadas a cada sub característica para evaluar la calidad del producto software y los niveles requeridos para alcanzar las necesidades del usuario. La tabla 4.50, presenta el plan de medición aplicado durante el desarrollo del producto software.

CARACTERISTICA	**SUB CARACTERISTICA**	**METRICAS**	**NIVEL REQUERIDO**
Funcionalidad	Aplicabilidad	MI , ME	70.00%
	Precisión	MI , ME	72.00%
Fiabilidad	Madurez	MI , ME	73.00%
	Tolerancia a fallas	MI , ME	50.00%
Usabilidad	Entendibilidad	MI , ME	68.00%
	Facilidad de aprendizaje	MI , ME	71.00%
	Operabilidad	MI , ME	65.00%
Eficiencia	Utilización de recursos	MI	60.00%
	Comportamiento en el tiempo	ME	50.00%
Facilidad de Mantenimiento	Cambiabilidad	MI	65.00%
Portabilidad	Adaptabilidad	MI	80.00%
	Instalabilidad	ME	60.00%

Tabla N° 4.49: Categorías de medición de calidad interna y externa

SUB CARACTERISTICA	ENTREGABLES A SER EVALUADOS	METRICAS INTERNAS USADAS	METRICAS EXTERNAS USADAS
Aplicabilidad	1. Catálogo de requisitos. 2. Descripción de casos de uso. 3. Tareas de ingeniería. 4. Interfaz gráfica de usuario. 5. Código fuente.	1. Adecuación funcional. 2. Integridad de implementación funcional. 3. Cobertura de la implementación funcional.	1. Adecuación funcional 2. Integridad de implementación funcional 3. Cobertura de implementación funcional
Precisión	1. Catálogo de requisitos. 2. Código fuente. 4. Pruebas unitarias. 5. Reporte de pruebas de aceptación.	1. Exactitud de cálculos.	1. Exactitud de cálculos.
Madurez	1. Reporte de pruebas unitarias. 2. Reporte de pruebas de integración. 3. Casos de pruebas de aceptación. 4. Reporte de pruebas de aceptación.	1. Eliminación de fallas.	1. Densidad de fallas contra los casos de prueba. 2. Madurez de la prueba
Tolerancia a fallas	1. Catálogo de requisitos. 2. Reporte de pruebas unitarias. 3. Casos de prueba de aceptación. 4. Reporte de pruebas de aceptación.	1. Prevención de operación incorrecta.	1. Prevención de operación incorrecta.
Entendibilidad	1. Catálogo de requisitos. 2. Descripción de casos de uso. 3. Tareas de ingeniería. 4. Interfaz gráfica de usuario.	1. Función de comprensión	1. Comprensión de entradas y salidas
Facilidad de aprendizaje	1. Catálogo de requisitos. 2. Lista de casos de uso. 3. Descripción de casos de uso. 4. Plan de versión. 5. Tareas de ingeniería.	1. Integridad de la documentación del usuario y/o facilidad de ayuda	1. Facilidad de aprender a realizar una tarea en uso
Operabilidad	1. Descripción de casos de uso. 2. Tareas de ingeniería. 3. Código fuente. 4. Interfaz gráfica de usuario.	1. Claridad de mensajes. 2. Claridad de la interfaz.	1. Entendibilidad del mensaje en uso
Utilización de recursos	1. Código fuente. 2. Estándares de codificación.	1. Utilización de memoria.	

Comportamiento en el tiempo	1. Casos de prueba. 2. Reporte de pruebas de aceptación. 3. Reporte de pruebas de integración.		1. Rendimiento
Cambiabilidad	1. Catálogo de requisitos. 2. Código fuente.	1. Registro de cambios	
Adaptabilidad	1. Catálogo de requisitos. 2. Arquitectura técnica (Diagrama de despliegue, diagrama de componentes).	1. Adaptabilidad al entorno organizacional (adaptabilidad a la organización y a la infraestructura de la misma)	
Instalabilidad	1. Procedimiento de instalación.		1. Facilidad de reinstalación

Tabla N° 4.50: Plan de medición

A. Resultados de la Medición de Calidad del Producto Software - ICONIX

La tabla 4.51 se ha construido aplicando el procedimiento descrito en el capítulo III, subtítulo 3.5.6, y las medidas del nivel obtenido para cada sub característica se muestran en el anexo D. La tabla 4.53 se ha construido para interpretar el valor de calidad del software, que indica el grado de calidad del producto software desarrollado con ICONIX, la escala fue establecida en base a experiencias de empresas desarrolladoras de Software.

Característica	Peso Característica	Sub Característica	Peso Sub Característica	Nivel Requerido	Nivel Obtenido
Funcionalidad	20.00%	Aplicabilidad	40.00%	70.00%	80.05%
		Precisión	60.00%	72.00%	71.46%
			100.00%	**71.20%**	**74.90%**
Fiabilidad	20.00%	Madurez	45.00%	73.00%	79.72%
		Tolerancia a Fallos	55.00%	50.00%	62.57%
			100.00%	**60.35%**	**70.29%**
Usabilidad	20.00%	Entendimiento	30.00%	68.00%	75.83%
		Aprendizaje	35.00%	71.00%	57.00%
		Operabilidad	35.00%	65.00%	67.92%
			100.00%	**68.00%**	**66.47%**
Eficiencia	10.00%	Utilización de Recursos	40.00%	60.00%	81.25%
		Comportamiento en el tiempo	60.00%	50.00%	68.18%
			100.00%	**54.00%**	**73.41%**
Facilidad de Mantenimiento	10.00%	Cambiabilidad	100.00%	65.00%	60.00%
			100.00%	**65.00%**	**60.00%**

Portabilidad	20.00%	Adaptabilidad	60.00%	80.00%	100.00%
		Instalabilidad	40.00%	60.00%	80.00%
	100.00%		**100.00%**	**72.00%**	**92.00%**
			TOTAL	**66.21%**	**74.07%**

Tabla N° 4.51: Base para evaluar la calidad del producto software - ICONIX

CARACTERISTICA	RESULTADO	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	Cumple	71.20%	74.90%
Fiabilidad	Cumple	60.35%	70.29%
Usabilidad	No cumple	68.00%	66.47%
Eficiencia	Cumple	54.00%	73.41%
Facilidad de Mantenimiento	No cumple	65.00%	60.00%
Portabilidad	Cumple	72.00%	92.00%
	Total	**66.21%**	**74.07%**

Tabla N° 4.52: Calificación de la calidad del producto software - ICONIX

RESULTADO	ESCALA
25%	Deficiente
60%	Insuficiente
80%	Satisfactorio
100%	Excede exigencia

Tabla N° 4.53: Escala para valorar la calidad del producto software - XP y ICONIX

RANGO	ESCALA PARA VALORAR
[0 – 22.35>	Deficiente
[22.99 - 66.21>	Insuficiente
[66.21 - 71.51>	Satisfactorio
[71.51 - 100]	Excede exigencia

Tabla N° 4.54: Rangos para interpretar la calidad del producto software - ICONIX

La tabla 4.54, se ha construido siguiendo el procedimiento descrito en el capítulo III, subtítulo 3.5.6, párrafo E, para interpretar el grado de calidad del producto software con el proceso ICONIX.

B. Resultados de la Medición de Calidad del Producto Software - XP

La tabla 4.55, se construyó igual que la tabla 4.51, el nivel obtenido calculado de cada sub característica se muestra en el anexo D.

Característica	Peso Característica	Sub Caracteris-tica	Peso Sub Característica	Nivel Requerido	Nivel Obtenido
Funcionalidad	20.00%	Aplicabilidad	40.00%	70.00%	81.52%
		Precisión	60.00%	72.00%	70.42%
			100.00%	**71.20%**	**74.86%**
Fiabilidad	20.00%	Madurez	45.00%	73.00%	72.92%
		Tolerancia a Fallos	55.00%	50.00%	50.53%
			100.00%	**60.35%**	**60.60%**
Usabilidad	20.00%	Entendimiento	30.00%	68.00%	73.81%
		Aprendizaje	35.00%	71.00%	71.25%
		Operabilidad	35.00%	65.00%	67.36%
			100.00%	**68.00%**	**70.66%**
Eficiencia	10.00%	Utilización de Recursos	40.00%	60.00%	66.67%
		Comportamiento en el tiempo	60.00%	50.00%	57.02%
			100.00%	**54.00%**	**60.88%**
Facilidad de Mantenimiento	10.00%	Cambiabilidad	100.00%	65.00%	66.67%
			100.00%	**65.00%**	**66.67%**
Portabilidad	20.00%	Adaptabilidad	60.00%	80.00%	100.00%
		instalabilidad	40.00%	60.00%	75.00%
	100.00%		**100.00%**	**72.00%**	**90.00%**
				TOTAL	**66.21%**	**71.98%**

Tabla N° 4.55: Base para evaluar la calidad del producto software - XP

CARACTERISTICA	RESULTADO	NIVEL REQUERIDO	NIVEL OBTENIDO
Funcionalidad	Cumple	71.20%	74.86%
Fiabilidad	Cumple	60.35%	60.60%
Usabilidad	Cumple	68.00%	70.66%
Eficiencia	Cumple	54.00%	60.88%
Facilidad de Mantenimiento	Cumple	65.00%	66.67%
Portabilidad	Cumple	72.00%	90.00%
	Total	**66.21%**	**71.98%**

Tabla N° 4.56: Calificación de la calidad del producto software - XP

La tabla 4.57 se ha construido de la misma forma que la tabla 4.54, para interpretar el grado de calidad del producto software con el proceso XP.

RANGO	ESCALA PARA VALORAR
[0 – 22.99>	Deficiente
[22.99 - 66.21>	Insuficiente
[66.21- 73.58>	Satisfactorio
[73.58- 100]	Excede exigencia

Tabla N° 4.57: Rangos para interpretar la calidad del producto software - XP

4.1.5 ANALISIS ESTADISTICO DE LA CALIDAD DEL PRODUCTO SOFTWARE

Según el procedimiento descrito en la sección 3.5.7 y los resultados del anexo D, realizamos el análisis estadístico para la calidad interna, calidad externa, calidad del producto con ICONIX y calidad de producto con XP.

A. Análisis de la Calidad Interna

Para determinar si existe diferencia significativa entre el nivel requerido y el nivel obtenido según sub características, realizamos las pruebas de igualdad de proporciones para las frecuencias porcentuales aplicando métricas internas – XP, ver la tabla 4.58. Planteamos las siguientes hipótesis estadísticas:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, calidad interna - XP.

Ha: Las proporciones del nivel obtenido, es superior a las proporciones del nivel requerido, calidad interna - XP.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	77,95%	1,45123	0,07336	0,0794872
	Precision	72,00%	62,50%	-1,77022	0,03835	-0,0950000
		71,20%	**68,68%**	-0,46570	0,32072	-0,0252051
Fiabilidad	Madurez	73,00%	66,67%	-1,19354	0,11633	-0,0633333
	Tolerancia a Fallos	50,00%	34,88%	-2,52944	0,00571	-0,1511628
		60,35%	**49,19%**	-1,90944	0,0281	-0,1116395
Usabilidad	Entendimiento	68,00%	66,67%	-0,23914	0,4055	-0,0133333
	Aprendizaje	71,00%	75,00%	0,73753	0,2304	0,0400000
	Operabilidad	65,00%	70,83%	1,02323	0,1531	0,0583333
		68,00%	**71,04%**	0,54555	0,29269	0,0304167
Eficiencia	Comportamiento en tiempo	60,00%	66,67%	0,80508	0,21039	0,0666667
		24,00%	**26,67%**	0,36939	0,35592	0,0266667
Mantenibilidad	No aplicable	65,00%	66,67%	0,20672	0,41811	0,0166667
		65,00%	**66,67%**	0,20672	0,41811	0,0166667
Portabilidad	Facilidad Instalación	80,00%	100,00%	4,18330	1,4E-05	0,2000000
		48,00%	**60,00%**	2,00959	0,02224	0,1200000
	% Total	**58,41%**	**59,11%**	0,26751	0,39454	0,0070477

Tabla N° 4.58: Pruebas de igualdad de proporciones según sub-características, calidad interna - XP

La tabla 4.59, muestra la distribución de frecuencias porcentuales del nivel requerido y del nivel obtenido, según características de evaluación del software usando métricas internas - XP.

Características	Nivel Requerido	Nivel Obtenido
Funcionabilidad	71,20%	68,68%
Fiabilidad	60,35%	49,19%
Usabilidad	68,00%	71,04%
Eficiencia	24,00%	26,67%
Mantenibilidad	65,00%	66,67%
Portabilidad	48,00%	60,00%
Proporción Total	**58,41%**	**59,11%**

Tabla N° 4.59: Distribución de frecuencias porcentuales según característica, calidad interna - XP

La tabla 4.60, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software, calidad interna - XP.

	Valor Teórico	Valor Muestral	gl	p-valor
Ji-cuadrado multinomial	15,09	13,9703241	5	0,016257

Tabla N° 4.60: Prueba múltiple de igualdad de proporciones según característica, calidad interna - XP

Para determinar si existe diferencia significativa entre el nivel requerido y el nivel obtenido según sub características, aplicando métricas internas – ICONIX, efectuamos la prueba como se observa en la tabla 4.61. Planteamos las siguientes hipótesis estadísticas:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, calidad interna - ICONIX.

Ha: Las proporciones del nivel obtenido, es superior a las proporciones del nivel requerido, calidad interna - ICONIX.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	73,48%	0,63624	0,26231	0,0348485
	Precision	72,00%	72,92%	0,17081	0,43219	0,0091667
		71,20%	**73,14%**	0,35917	0,35973	0,0194394
Fiabilidad	Madurez	73,00%	75,00%	0,37691	0,35312	0,0200000
	Tolerancia a Fallos	50,00%	53,70%	0,61975	0,26771	0,0370370
		60,35%	**63,29%**	0,50234	0,30771	0,0293704
Usabilidad	Entendimiento	68,00%	66,67%	-0,23914	0,4055	-0,0133333
	Aprendizaje	71,00%	50,00%	-3,87204	5,4E-05	-0,2100000
	Operabilidad	65,00%	69,17%	0,73088	0,23243	0,0416667
		68,00%	**61,71%**	-1,12846	0,12956	-0,0629167
Eficiencia	Comportami-	60,00%	81,25%	2,56618	0,00514	0,2125000

	to en tiempo					
		24,00%	**32,50%**	1,17744	0,11951	0,0850000
Mantenibilidad	Cambiabilidad	65,00%	60,00%	-0,62017	0,26757	-0,0500000
		65,00%	**60,00%**	-0,62017	0,26757	-0,0500000
Portabilidad	Facilidad de Instalación	80,00%	100,00%	4,18330	1,4E-05	0,2000000
		48,00%	**60,00%**	2,00959	0,02224	0,1200000
	% Total	**58,41%**	**60,88%**	0,93674	0,17445	0,0246786

Tabla N° 4.61: Pruebas de igualdad de proporciones según sub-características, calidad interna - ICONIX

La tabla 4.62, muestra la distribución de frecuencias porcentuales del nivel requerido y del nivel obtenido, según características de evaluación del software, calidad interna - ICONIX.

Características	**Nivel Requerido**	**Nivel Obtenido**
Funcionabilidad	71,20%	73,14%
Fiabilidad	60,35%	63,29%
Usabilidad	68,00%	61,71%
Eficiencia	24,00%	32,50%
Mantenibilidad	65,00%	60,00%
Portabilidad	48,00%	60,00%
Proporción Total	**58,41%**	**60,88%**

Tabla N° 4.62: Distribución de frecuencias porcentuales según característica, calidad interna - ICONIX

La tabla 4.63, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software, calidad interna - ICONIX.

	Valor Teórico	**Valor Muestral**	**gl**	**p-valor**
Ji-cuadrado multinomial	15,09	15,011552	5	0,010313

Tabla N° 4.63: Prueba múltiple de igualdad de proporciones según característica, calidad interna - ICONIX

B. Análisis de la Calidad Externa

La tabla 4.64 muestra la prueba a nivel de sub características. Planteamos las siguientes hipótesis estadísticas:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, calidad externa - XP.

Ha: Las proporciones del nivel obtenido, es superior a las proporciones del nivel requerido, calidad externa - XP.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	85,09%	2,75582	0,00293	0,1509425
	Precision	72,00%	78,33%	1,18015	0,11897	0,0633333
		71,20%	**81,04%**	1,81763	0,03456	0,0983770
Fiabilidad	Madurez	73,00%	79,17%	1,16213	0,12259	0,0616667
	Tolerancia a Fallos	50,00%	66,67%	2,78887	0,00265	0,1666667
		60,35%	**72,29%**	2,04246	0,02055	0,1194167
Usabilidad	Entendimiento	68,00%	80,95%	2,35711	0,00909	0,1295238
	Aprendizaje	71,00%	67,50%	-0,64534	0,25935	-0,0350000
	Operabilidad	65,00%	63,89%	-0,19490	0,42274	-0,0111111
		68,00%	**70,27%**	0,40747	0,34183	0,0227183
Eficiencia	Comportami-ento tiempo	50,00%	57,02%	0,83033	0,20318	0,0701754
		30,00%	**34,21%**	0,54358	0,29337	0,0421053
Mantenibilidad	No aplicable	0,00%	0,00%	-	-	0,0000000
		0,00%	**0,00%**	-	-	0,0000000
Portabilidad	Facilidad Instalación	60,00%	75,00%	2,56174	0,00521	0,1500000
		24,00%	**30,00%**	1,17541	0,11992	0,0600000
	% Total	**47,71%**	**54,14%**	2,40890	0,00800	0,0643129

Tabla N° 4.64: Pruebas de igualdad de proporciones según sub-características, calidad externa - XP

Tabla 4.65, distribución de frecuencias porcentuales nivel requerido y nivel obtenido, según características de evaluación software, calidad externa - XP.

Características	Nivel Requerido	Nivel Obtenido
Funcionabilidad	71,0%	81,0%
Fiabilidad	60,4%	72,3%
Usabilidad	68,0%	70,3%
Eficiencia	30,0%	34,2%
Mantenibilidad	0,0%	0,0%
Portabilidad	24,0%	30,0%
Proporción Total	**47,71%**	**54,14%**

Tabla N° 4.65: Distribución de frecuencias porcentuales según característica, calidad externa - XP

La tabla 4.66, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software, calidad externa - XP.

	Valor Teórico	Valor Muestral	gl	p-valor
Ji-cuadrado multinomial	15,09	20,78384418	5	0,000891

Tabla N° 4.66: Prueba múltiple de igualdad de proporciones según característica, calidad externa - XP

La tabla 4.67, muestra la prueba a nivel de sub características. Planteamos las siguientes hipótesis estadísticas:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, calidad externa - ICONIX.

Ha: Las proporciones del nivel obtenido, es superior a las proporciones del nivel requerido, calidad externa - ICONIX.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	86,62%	3,03445	0,00121	0,1662037
	Precisión	72,00%	70,00%	-0,37268	0,64531	-0,0200000
		71,20%	**76,65%**	1,00661	0,15706	0,0544815
Fiabilidad	Madurez	73,00%	84,44%	2,38567	0,00852	0,1144444
	Tolerancia a Fallos	50,00%	71,43%	3,58569	0,00017	0,2142857
		60,35%	**77,29%**	2,89663	0,00189	0,1693571
Usabilidad	Entendimiento	68,00%	85,00%	3,04908	0,00115	0,1700000
	Aprendizaje	71,00%	64,00%	-1,29068	0,09841	-0,0700000
	Operabilidad	65,00%	66,67%	0,29235	0,38501	0,0166667
		68,00%	**71,23%**	0,57992	0,28098	0,0323333
Eficiencia	Comportami-ento tiempo	50,00%	68,18%	2,45130	0,00711	0,1818182
		30,00%	**40,91%**	1,40836	0,07951	0,1090909
Mantenibilidad	No aplicable	0,00%	0,00%			0,0000000
		0,00%	**0,00%**			0,0000000
Portabilidad	Facilidad de Instalación	60,00%	80,00%	0,07951	0,46831	0,2000000
		24,00%	**32,00%**	1,56721	0,05853	0,0800000
	% Total	**47,71%**	**55,52%**	2,92693	0,00172	0,0781435

Tabla N° 4.67: Pruebas de igualdad de proporciones según sub-características, calidad externa - ICONIX

La tabla 4.68, muestra la distribución de frecuencias porcentuales del nivel requerido y del nivel obtenido, según características de evaluación del software, calidad externa - ICONIX.

Característica	Nivel Requerido	Nivel Obtenido
Funcionabilidad	71,2%	76,6%
Fiabilidad	60,4%	77,3%
Usabilidad	68,0%	71,2%
Eficiencia	30,0%	40,9%
Mantenibilidad	0,0%	0,0%
Portabilidad	24,0%	32,0%
Proporción Total	**47,71%**	**55,52%**

Tabla N° 4.68: Distribución de frecuencias porcentuales según característica, calidad externa - ICONIX

La tabla 4.69, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software, calidad externa - ICONIX.

	Valor Teórico	Valor Muestral	gl	p-valor
Ji-cuadrado multinomial	15,09	41,84887247	5	0,000000

Tabla N° 4.69: Prueba múltiple de igualdad de proporciones según característica, calidad externa - ICONIX

C. Análisis de la Calidad del Producto - ICONIX

Para determinar si existe diferencia significativa entre el nivel requerido y el nivel obtenido según sub características de evaluación del software basado en la calidad del producto – ICONIX, ver tabla 4.71. Planteamos la siguiente hipótesis estadística:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, basado en la calidad del producto - ICONIX.
Ha: Las proporciones del nivel obtenido es superior a las proporciones del nivel requerido, basado en la calidad del producto - ICONIX.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	80,05%	2,59552	0,00472	0,1005242
	Precision	72,00%	71,46%	-0,11943	0,45247	-0,0054167
		71,20%	**74,90%**	0,81619	0,20720	0,0369597
Fiabilidad	Madurez	73,00%	79,72%	1,39552	0,08143	0,0672000
	Tolerancia a Fallos	50,00%	62,57%	3,12912	0,00088	0,1256685
		60,35%	**70,29%**	2,03115	0,02112	0,0993577
Usabilidad	Entendimiento	68,00%	75,83%	1,40497	0,08002	0,0783333
	Aprendizaje	71,00%	57,00%	-2,02318	0,02153	-0,1400000
	Operabilidad	65,00%	67,92%	0,34066	0,36668	0,0291833
		68,00%	**66,47%**	-0,32769	0,37157	-0,0152858
Eficiencia	Utilización Recursos	60,00%	81,25%	4,06907	0,00002	0,2125000
	Comportami-ento tiempo	50,00%	68,18%	3,39177	0,00035	0,1818182
		54,00%	**73,41%**	2,75369	0,00295	0,1940909
Mantenibilidad	Cambiabilidad	65,00%	60,00%	-0,84515	0,19901	-0,0500000
		65,00%	**60,00%**	-0,84515	0,19901	-0,0500000
Portabilidad	Adaptabilidad	80,00%	100,00%	4,18330	0,00001	0,2000000
	Instalabilidad	60,00%	80,00%	3,65148	0,00013	0,2000000
		72,00%	**92,00%**	4,45435	0,00000	0,2000000

	% Total	**66,21%**	**74,07%**	3,71652	0,00010	0,0786154

Tabla N° 4.70: Pruebas de igualdad de proporciones según sub-características, basado en la calidad del producto - ICONIX

La tabla 4.71, muestra la distribución de frecuencias porcentuales del nivel requerido y del nivel obtenido, según características de evaluación del software basado en la calidad del producto - ICONIX.

Características	**Nivel Requerido**	**Nivel Obtenido**
Funcionabilidad	71,20%	74,90%
Fiabilidad	60,35%	70,29%
Usabilidad	68,00%	66,47%
Eficiencia	54,00%	73,41%
Mantenibilidad	65,00%	60,00%
Portabilidad	72,00%	92,00%
Proporción Total	**66,21%**	**74,07%**

Tabla N° 4.71: Distribución de frecuencias porcentuales según característica, basado en la calidad del producto - ICONIX

La tabla 4.72, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software basado en la calidad del producto - ICONIX.

	Valor Teórico	**Valor Muestral**	**gl**	**p-valor**
Ji-cuadrado multinomial	15,09	24,10702	5	0,000217

Tabla N° 4.72: Prueba múltiple de igualdad de proporciones según característica, basado en la calidad del producto - ICONIX

D. Análisis de la Calidad del Producto - XP

Para determinar si existe diferencia significativa entre el nivel requerido y el nivel obtenido según sub características de evaluación del software basado en la calidad del producto – XP, ver tabla 4.73. Planteamos la siguiente hipótesis estadística:

Ho: Las proporciones del nivel obtenido es igual a las proporciones del nivel requerido, basado en la calidad del producto - XP.

Ha: Las proporciones del nivel obtenido es superior a las proporciones del nivel requerido, basado en la calidad del producto - XP.

Característica	Sub Característica	Nivel Requerido	Nivel Obtenido	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	70,00%	81,52%	2,97429	0,00147	0,1151936
	Precision	72,00%	70,42%	-0,36344	0,35814	-0,0158500
		71,20%	**74,86%**	0,80753	0,20968	0,0365674
Fiabilidad	Madurez	73,00%	72,92%	-0,01885	0,49248	-0,0008167
	Tolerancia a Fallos	50,00%	50,53%	0,07815	0,46885	0,0052686
		60,35%	**60,60%**	0,05172	0,47938	0,0025302
Usabilidad	Entendimiento	68,00%	73,81%	1,04177	0,14876	0,0580833
	Aprendizaje	71,00%	71,25%	0,03259	0,48700	0,0025000
	Operabilidad	65,00%	67,36%	0,29293	0,38479	0,0236167
		68,00%	**70,66%**	0,56950	0,28451	0,0265658
Eficiencia	Utilización Recurso	60,00%	66,67%	1,27657	0,10088	0,0666667
	Comport. Tiempos	50,00%	57,02%	1,30911	0,09525	0,0701754
		54,00%	**60,88%**	1,11248	0,13297	0,0687719
Mantenibilidad	Cambiabilidad	65,00%	66,67%	0,24708	0,40242	0,0166667
		65,00%	**66,67%**	0,24708	0,40242	0,0166667
Portabilidad	Adaptabilidad	80,00%	100,00%	3,87298	0,00005	0,2000000
	Instalabilidad	60,00%	75,00%	2,73861	0,00309	0,1500000
		72,00%	**90,00%**	4,00892	0,00003	0,1800000
	% Total	**66,21%**	**71,98%**	2,72664	0,00320	0,0576766

Tabla N° 4.73: Prueba de igualdad de proporciones según sub-características, basado en la calidad del producto - XP

La tabla 4.74, muestra la distribución de frecuencias porcentuales del nivel requerido y del nivel obtenido, según características de evaluación del software basado en la calidad del producto - XP.

Características	Nivel Requerido	Nivel Obtenido
Funcionabilidad	71,20%	74,86%
Fiabilidad	60,35%	60,60%
Usabilidad	68,00%	70,66%
Eficiencia	54,00%	60,88%
Mantenibilidad	65,00%	66,67%
Portabilidad	72,00%	90,00%
Proporción Total	**66,21%**	**71,98%**

Tabla N° 4.74: Distribución de frecuencias porcentuales según característica, basado en la calidad del producto - XP

La tabla 4.75, muestra la prueba múltiple de igualdad de proporciones para las frecuencias porcentuales del nivel requerido y nivel obtenido, según características, para la evaluación del software basado en la calidad del producto - XP.

	Valor Teórico	Valor muestral	gl	p-valor
Ji-cuadrado multinomial	15,09	19,98932364	5	0,001256

Tabla N° 4.75: Prueba múltiple de igualdad de proporciones según característica, basado en la calidad del producto - XP

4.2 DISCUSION DE RESULTADOS

Según la tabla 4.58, para las sub características que tienen asociadas un p-valor menor que 0,01 rechazamos la hipótesis nula de igualdad de proporciones. Por tanto, para la sub característica "tolerancia a fallos-fiabilidad" el nivel obtenido es inferior al nivel requerido, para la sub característica "facilidad de instalación-portabilidad" existe diferencia significativa para el nivel obtenido el que es superior al nivel requerido. Entonces, a nivel de sub características evaluando la calidad interna del software para el proceso XP, tenemos que 10% de sub características superan significativamente el nivel requerido, 10% de sub características son inferiores al nivel requerido y 80% de sub-características son iguales al nivel requerido.

La tabla 4.60 muestra que el p-valor asociado a la prueba Chi-cuadrado multinomial (13,9703241) es de 0,016257, lo que indica que aceptamos la hipótesis nula, porque el porcentaje real de cometer un error tipo I es aproximadamente .1,6%. Entonces, concluimos que la evaluación de la calidad interna del software, desarrollado con XP, demuestra que el nivel obtenido es estadísticamente igual al nivel requerido.

En la tabla 4.61 para las sub-características que tengan asociadas un p-valor menor de 0,01 rechazamos la hipótesis nula de igualdad de proporciones. Entonces, para la sub característica "aprendizaje-usabilidad" el nivel obtenido es inferior estadísticamente al nivel requerido, para las sub característica "comportamiento en tiempos-eficiencia" y "facilidad de instalación-portabilidad" existen diferencias significativas entre el nivel obtenido que es superior al nivel requerido. Por tanto, a nivel de sub características evaluando la calidad interna del software desarrollado con la metodología ICONIX, tenemos que 20% de sub características superan significativamente al nivel requerido y 10% de sub características son inferiores al nivel requerido, 70% restante de sub características son estadísticamente iguales al nivel requerido.

De acuerdo a la tabla 4.63, el p-valor asociado a la prueba Chi-cuadrado multinomial (15,011552) es 0,010313, lo cual nos indica que se acepta la hipótesis nula, porque el porcentaje real de cometer un error tipo I es aproximadamente 1%. Donde, concluimos que la evaluación de la calidad interna del software desarrollado aplicando ICONIX, demuestra que el nivel obtenido es estadísticamente igual al nivel requerido.

A partir de los análisis anteriores, acerca de la evaluación de la calidad interna del software aplicando XP y ICONIX, no se establece mayor diferencia significativa de las sub características y características, ver las tablas 4.58 y 4.51. Por tanto, de la evidencia estadística indicamos que la calidad interna del software desarrollado con ICONIX y XP son estadísticamente iguales para el nivel obtenido.

De acuerdo a la tabla 4.64, para las sub características que tienen asociados un p-valor menor a 0,01, rechazamos la hipótesis nula de igualdad de proporciones. Entonces, para las sub características "aplicabilidad-funcionalidad", "tolerancia a fallos-fiabilidad", "entendimiento-usabilidad" y "facilidad de instalación-portabilidad" existe diferencia significativa entre las proporciones del nivel requerido y nivel obtenido. Por tanto, a nivel de sub características evaluando la calidad externa del software desarrollado con XP, 40% de sub características superan significativamente al nivel requerido y 60% de sub características son estadísticamente iguales al nivel requerido.

Según la tabla 4.66, el p-valor asociado a la prueba Chi-cuadrado multinomial (20,78384418) es de 0,000891, lo cual indica que rechazamos la hipótesis nula y aceptamos la hipótesis alternativa, porque el porcentaje real de cometer un error tipo I es aproximadamente 0,09%. Donde, concluimos que la evaluación de la calidad externa del software desarrollado con XP demuestra que el nivel obtenido es estadísticamente superior al nivel requerido.

La tabla 4.67, muestra que para las sub características que tengan asociadas un p-valor menor a 0,01 rechazamos la hipótesis nula de igualdad de proporciones. Entonces, para la sub característica "aplicabilidad-

funcionalidad", "madurez, tolerancia a fallos-fiabilidad", "entendimiento-usabilidad" y, "comportamiento en el tiempo-eficiencia" existe diferencia significativa entre las proporciones del nivel requerido y nivel obtenido. Por tanto, a nivel de sub características evaluando la calidad externa del software desarrollado con ICONIX, 50% de sub características superan significativamente al nivel requerido y, 50% restante de sub características son estadísticamente iguales al nivel requerido.

En relación a la tabla 4.69, observamos que el p-valor asociado a la prueba Chi-cuadrado (41,84887247) es de 0,000000 que indica rechazar la hipótesis nula y aceptar la hipótesis alternativa, porque el porcentaje real de cometer un error tipo I es 0,00%. Concluimos que la evaluación de la calidad externa del software desarrollado con ICONIX demuestra que el nivel obtenido es estadísticamente superior al nivel requerido.

De ambos análisis de la evaluación de la calidad externa del software desarrollado con XP y ICONIX, observamos que donde se establece mayor diferencia significativa de las sub-características, y características es con la metodología ICONIX, ver las tablas 4.64 y 4.67. Por tanto, de la evidencia estadística indicamos que la metodología ICONIX para la calidad externa es estadísticamente superior a la metodología XP.

De la tabla 4.73, observamos que para las sub características que tienen asociadas un p-valor menor a 0,01 rechazamos la hipótesis nula de igualdad de proporciones. Entonces, para las sub características "aplicabilidad-funcionalidad" y "adaptabilidad, instalabilidad-portabilidad" el nivel obtenido es superior estadísticamente al nivel requerido. Por tanto, a nivel de sub características en la evaluación de la calidad del producto software desarrollado con XP, 17% de sub características superan significativamente al nivel requerido y 83% de sub características son estadísticamente iguales al nivel requerido.

En la tabla 4.75 observamos que el p-valor asociada a la prueba Chi-cuadrado multinomial (19,98932364) es de 0,001256 que indica rechazar la

hipótesis nula, porque el porcentaje real de cometer un error tipo I es aproximadamente 0,13%. Demostramos que al evaluar la calidad del producto software desarrollado con XP, el nivel obtenido es estadísticamente superior al nivel requerido.

El objetivo de XP esta dirigido a grupos pequeños y medianos de construcción de software, con requisitos poco definidos que cambian rápidamente o son de alto riesgo, las prácticas de XP están encaminadas a producir software de calidad. (Ariel y Bastarrica, 2005). La tabla 4.58 muestra que el software desarrollado aplicando las prácticas XP, para la calidad interna a nivel de sub características son, 10% inferiores, 10% superiores y 80% iguales respectivamente al nivel requerido, indicándonos que aplicar la practicas XP produce software con calidad interna.

En XP el código es el principal artefacto y el aseguramiento de la calidad se centra en el código con la programación por pares, la refactorización, las pruebas, el aseguramiento de la calidad de las historias de usuario se logra con la participación activa del cliente. (Ariel y Bastarrica, 2005). El plan de medición de la calidad interna y externa tabla 4.50, para las sub características: apliçabilidad, precisión, operabilidad, utilización de recurso y cambiabilidad utiliza el código fuente para la evaluación de la calidad del producto. Respecto a la madurez, comportamiento en el tiempo y, tolerancia a fallas utiliza casos de pruebas, pruebas unitarias, pruebas de integración, pruebas de aceptación, reporte de pruebas aceptación e integración. De acuerdo con la tabla 4.58, la única sub característica que presenta menor calidad que la requerida es la tolerancia a fallas, siendo en los demás casos superior o igual al nivel de calidad requerido.

Una encuesta de cuarenta y cinco preguntas sobre XP, informan que existe riesgo alto en la falta frecuente del cliente y programadores que no trabajan en pareja. Los aspectos positivos son la propiedad colectiva del código, programación en pareja y enfoque de XP sobre objetivos del cliente, sobre la pregunta si usaría otra vez XP, el 93.3 % responde que si y 6.7 % requiere que se mejore (Rumpe y Schröder, 2001). El desarrollo del software para comercializar tara aplicando XP ha tenido la colaboración de los actores directos

(productor, transformador, acopiador, consumidor) e indirectos (proveedores), formulando y revisando historias de usuario, probando las interfaces y evaluado el producto software, lo cual significa que el cliente ha participo activamente y se desarrollo en función a los objetivos del cliente, al evaluar la calidad del producto software, como se observa en la tabla 4.75, el nivel obtenido es superior al nivel requerido, resultando un software de calidad.

Las prácticas de desarrollo de software con XP obtienen el máximo rendimiento con equipos pequeños de programadores, soporte mínimo de procedimientos y documentación, se debe completar algunas prácticas de XP con procedimientos "ligeros" de control y documentación, denominados micro-prácticas, para aumentar el grado de confiabilidad del proceso XP (Sicilia et al., 2002). Las micro prácticas usadas fueron la estructuración de las historias de usuario mostradas en las tablas 4.9, 4.10 y 4.14, la arquitectura técnica figura 4.22, para mejorar la calidad del producto software que apoya la comercialización de la tara.

Los aspectos positivos de XP como las pruebas unitarias del código como práctica y factor clave para obtener software de alta calidad, integración continua para evitar problemas por defectos no detectados a tiempo y, los aspectos controversiales como: XP desalienta el diseño, es débil en documentación, no pasa para proyectos con seguridad crítica (González y Fernández, 2006; ápud Aiken, 2004). Hasta ahora no existen estudios cuantitativos que midan el grado de éxito obtenido al adoptar este paradigma (González y Fernández, 2006). La investigación persigue obtener un software de calidad para comercializar tara en la Región Ayacucho, según opinión de expertos la calidad del producto software es satisfactorio como se observa en las tablas 4.56 y 4.57, la evaluación cuantitativa por aplicar XP se muestra en la tabla 4.75, para la calidad del producto software sobre características y sub características, el nivel obtenido es superior al nivel requerido, obteniéndose un software de calidad, que aporta luces a la afirmación de (González y Fernández, 2006).

En la tabla 4.70 observamos que para las sub características que tienen

asociadas un p-valor menor a 0,01 rechazamos la hipótesis nula de igualdad de proporciones. Entonces, para las sub características: "aplicabilidad-funcionalidad", "tolerancia a fallos-fiabilidad" "utilización de recursos, comportamiento en el tiempo-eficiencia", "adaptabilidad, instalabilidad-portabilidad" el nivel obtenido es superior estadísticamente al nivel requerido. Por tanto, a nivel de sub características en la evaluación de la calidad del producto software desarrollado con ICONIX, 50% de sub características superan significativamente al nivel requerido, 50% de sub características restantes son estadísticamente iguales al nivel requerido.

En la tabla 4.72, observamos que el p-valor asociada a la prueba Chi-cuadrado multinomial (24,10702) es 0,000217, indica rechazar la hipótesis nula, porque el porcentaje real de cometer un error tipo I es aproximadamente 0,027%. Al evaluar la calidad del producto software desarrollado con ICONIX, el nivel obtenido es estadísticamente superior al nivel requerido.

Característica	Sub Característica	Nivel Obtenido XP	Nivel Obtenido ICONIX	Z aprox. binomial	p-valor	Diferencia proporciones
Funcionalidad	Aplicabilidad	81,52%	80.05%	-0.44739	0.32730	-0.0146758
	Precisión	70,42%	71.46%	0.22522	0.41090	0.0103833
		74,86%	**74.90%**	0.00829	0.49669	0.0003597
Fiabilidad	Madurez	72,92%	79.72%	1.71087	0.04355	0.0680000
	Tolerancia a Fallos	50,53%	62.57%	2.99732	0.00136	0.1203685
		60,60%	**70.29%**	1.98221	0.02373	0.0968577
Usabilidad	Entendimiento	73,81%	75.83%	0.38503	0.35011	0.0202333
	Aprendizaje	71,25%	57.00%	-2.06461	0.01948	-0.1425000
	Operabilidad	67,36%	67.92%	0.06630	0.47359	0.0055833
		70,66%	**66.47%**	-0.91992	0.17881	-0.0418858
Eficiencia	Utilización de Recursos	66,67%	81.25%	2.90146	0.00186	0.1458000
	Comportamiento en el tiempo	57,02%	68.18%	2.10304	0.01773	0.1116182
		60,88%	**73.41%**	1.81538	0.03473	0.1252909
Mantenibilidad	Cambiabilidad	66,67%	60.00%	-1.81754	0.03457	-0.0667000
		66,67%	**60.00%**	-1.81754	0.03457	-0.0667000
Portabilidad	Adaptabilidad	100,00%	100.00%	------	----	0.0000000
	Instalabilidad	75,00%	80.00%	1.62070	0.05254	0.0500000
		90,00%	**92.00%**	0.66667	0.25249	0.0200000
	% Total	**71,98%**	**74.07%**	1.39644	0.08129	0.0209154

Tabla N° 4.76: Prueba multinomial de igualdad de proporciones según característica y sub característica, basado en la calidad del producto de ICONIX y XP

En la tabla 4.76, observamos todas las características y sub-características asociadas a la calidad del producto de ICONIX y XP. La evidencia estadística indica que no existe diferencia significativa en la calidad del producto desarrollado con ICONIX y XP, porque para el porcentaje total comparativo obtenemos un p-valor = 0.08129 (superior a 0,05 e incluso a 0.01 de error tipo I), por tanto no podemos rechazar la hipótesis nula de igualdad de proporciones. A nivel de las sub-características "tolerancia a fallos, y utilización de recursos," el nivel obtenido para ICONIX es superior estadísticamente al nivel obtenido para XP, porque sus valores p asociados son menores que 0.01de error tipo I.

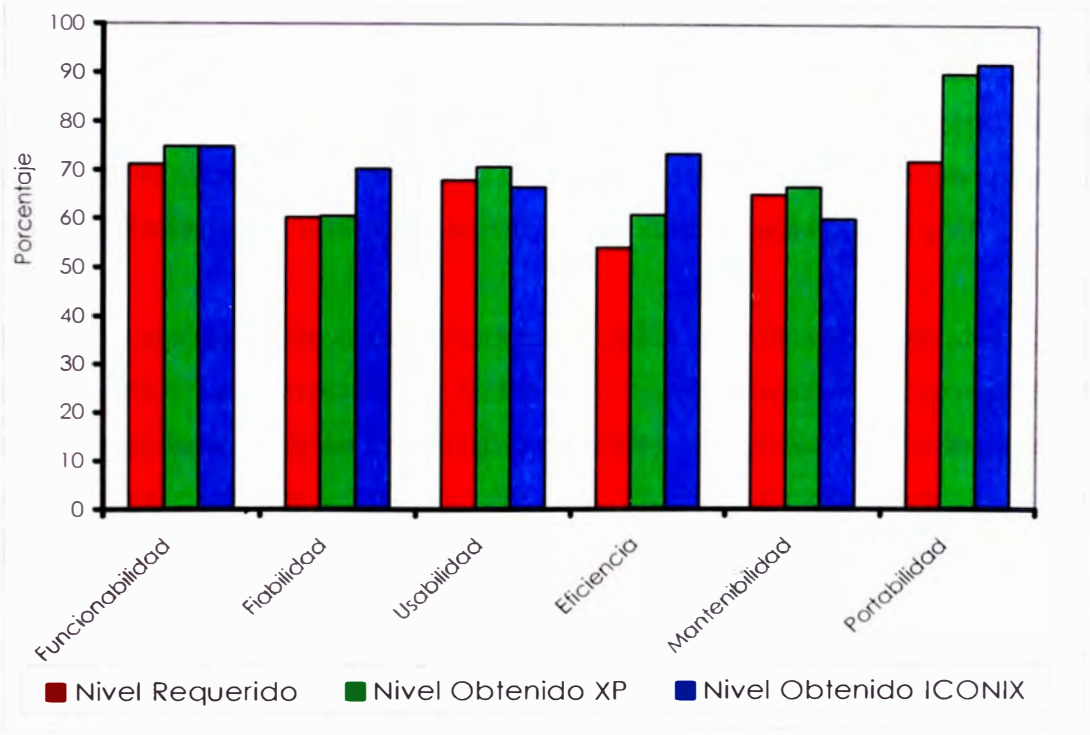

Figura N° 4.27: Comparación del nivel requerido y nivel obtenido según características basado en la calidad del producto aplicando ICONIX y XP.

El proceso ICONIX presenta un modelo estático que es refinado durante las iteraciones del modelo dinámico (Pantoja, 2009). Construimos el proceso de desarrollo de software usando ICONIX, descrito en las tablas 3.6 a 3.13, donde observamos las fases e hitos, particularmente los hitos que son la revisión y refinamiento a una fase, mejorando e incrementado el desarrollo del software durante la interacción entre el modelo estático y dinámico.

ICONIX es considerado como una metodología pura, práctica y simple, que

realiza análisis y diseño, con la característica "rastreabilidad de los requisitos", compuesto por los entregables: modelo de dominio, modelo de caso de uso, diagrama de robustez, diagrama de secuencia y diagrama de clase. (Pantoja, 2009). Los resultados mostrados en las figuras 4.1 a 4.21, tablas 4.1 a 4.8 y el anexo A, presentan los modelos y código producido, los entregables se han incrementado por la necesidad de evaluar la calidad del producto software desarrollado.

El modelo de calidad del software debe proporcionar una definición formal de conceptos acordados que aseguren la interpretación correcta del conocimiento (Martin, 2004). El estudio usa la NTP-ISO/IEC 9126-1:2004 modelo de calidad, NTP-ISO/IEC-TR 9126-2:2004 métricas externas, NTP-ISO/IEC-TR 9126-3:2005 métricas internas, NTP-ISO/IEC 14598-1:2005 evaluación del producto software, asegurando la correcta evaluación e interpretación del grado de calidad del producto software, desarrollado usando las metodologías ICONIX y XP.

Existen varios modelos de calidad para procesos y productos software, la mayor parte basados en la norma ISO 9126, las métricas validadas teóricamente son 3 % y validadas empíricamente 37 % (Esparza, 2006). De acuerdo al párrafo anterior, el modelo, las métricas y proceso de evaluación esta de acuerdo a normas estandarizadas, garantizando la confiabilidad de los resultados obtenidos.

El proceso de gestión de calidad del software, tiene tareas importantes como: control de calidad del software (probar el software para encontrar errores), asegurar la calidad del software (evaluar la calidad del software usando métricas) (Scalone, 2003). Las métricas son seleccionadas en función a los objetivos para comercializar tara, ver anexo E, los desarrolladores tienen objetivos como respetar el presupuesto, plazos, minimizar los errores antes y después de la entrega del producto e intentar mejorar la calidad y la productividad. El estudio no desarrollo la gestión completa de la calidad del software, porque sus objetivos son comparar dos procesos de desarrollo como son ICONIX y XP, las métricas se seleccionaron de acuerdo a las necesidades de los actores directos e indirectos (usuarios), observar las tablas 4.49 y 4.50.

Al asegurar la calidad del software, obtenemos beneficios como: menos errores, reducción del tiempo de pruebas y mantenimiento, mayor fiabilidad, lo negativo es difícil de aplicar en organizaciones pequeñas por la escasa disponibilidad de recursos para realizar las tareas (Diez, 2003). Los anexos C y D, sobre las métricas utilizadas y las medidas para la calidad interna, calidad externa y calidad del producto software, muestran objetivamente el aseguramiento de la calidad del producto software desarrollado con ICONIX y XP, por otra parte, interpretando las tablas 4.72 y 4.75, los procesos ICONIX y XP, producen software con mejor nivel que el requerido.

En estudio sobre la calidad del desarrollo de sistemas, los encuestados opinan que medir la calidad del producto software y las características de calidad como funcionalidad y usabilidad son las más importantes (Santana, 2003). La figura 4.27, muestra que para la funcionalidad el nivel obtenido es superior al nivel requerido usando los dos procesos, la usabilidad con XP supera el nivel requerido, para ICONIX estadísticamente los niveles requerido y obtenido son iguales, ratificando lo afirmado en la encuesta.

Una encuesta a 45 empresas, programadores y líderes de proyectos, resulta que el 84% opina que el aseguramiento de la calidad de sus productos es muy importante, de éstos, 57% tiene algún procedimiento establecido y 15% tiene una metodología específica, las características de calidad más importantes son la fiabilidad y usabilidad, las menos importantes portabilidad y facilidad de interoperación del producto final (Díaz, 2001). El aseguramiento de la calidad se realizó usando los estándares NTP-ISO/IEC TR 9126-1, NTP-ISO/IEC-TR 9126-3 y NTP-ISO/IEC-TR 9126-2, mostrados en anexo C y la medición de la calidad del producto software ver anexo D, que garantizan un procedimiento adecuado, de acuerdo a la prueba múltiple de igualdad de proporciones según característica, basado en la calidad del producto usando XP ver tabla 4.75 y usando ICONIX ver tabla 4.72, interpretamos que para todas las características el nivel obtenido supera al nivel requerido.

Para cada desarrollo de software, existe la posibilidad de aplicar diversas metodologías, los desarrolladores deben analizar la metodología que mejor se

adecue a sus necesidades del desarrollo y a los usuarios (Rafael, 2004). Siendo la comercialización de la tara en la Región Ayacucho un software pequeño o mediano, **donde la solución necesita un diseño disciplinado, entrega urgente, usando recursos limitados,** se selecciono el proceso ICONIX descrito en las tablas 3.6 a 3.13 y el proceso XP en tablas 3.14 a 3.16.

Para realizar proyectos de mediano tamaño, surgió un proceso que puede ser considerado de mediano tamaño (no grande como RUP ni pequeño como XP), es ICONIX. Siendo proceso intermedio y equilibrado, dirigido por casos de uso, como el RUP, que no tiene sobrecarga de modelado del RUP (Kitzberger, 2005). Analizando la evaluación de la calidad del producto software desarrollado con el proceso ágil XP y el proceso ICONIX, establecemos que existe mejor diferencia estadística de sub características y características con ICONIX, ver tablas 4.70, 4.73 y figura 4.27.

Para desarrollar software de calidad elegimos una metodología para un equipo y un proyecto, los métodos ágiles ponen vital importancia a los cambios rápidos durante el desarrollo, confianza en habilidades del equipo y mantener una buena relación con el cliente (Figueroa et al., 2007). Luego de evaluar y comparar el nivel requerido y obtenido del producto software desarrollado con ICONIX y XP, según las tablas 4.51 y 4.55, se verifica la afirmación de referencia.

Desarrollar software usando métodos ágiles con calidad de producto tiene limitaciones y ventajas como: ningún método define la administración de requisitos, la medición y análisis de calidad no esta definida, la planificación se aplica a iteraciones cortas, solo se gestiona la configuración del código (Ariel y Bastarrica, 2005). El proceso ágil XP descrito en tablas 3.14 a 3.16 no presenta la administración de requisitos, la evaluación de la calidad no esta definida pero tiene importancia las pruebas unitarias realizadas con Junit, reporte de pruebas de integración tabla 4.40, casos de prueba de aceptación tabla 4.24, reporte de casos de prueba de aceptación tabla 4.41 a 4.46, reporte de pruebas unitarias tabla 4.39, plan de alto nivel tabla 4.10, plan de versión tabla 4.14 y plan de iteración tabla 4.23, la calidad del producto software pasa los niveles requeridos mediante la aplicación correcta del proceso XP.

Las metodologías ágiles ofrecen solución a proyectos pequeños y medianos, por otra parte existe escasez de estudios sobre aspectos teóricos y prácticos de su uso. La investigación en metodologías ágiles, está orientada hacia líneas como: métricas y evaluación del proceso, herramientas específicas para apoyar las prácticas ágiles, aspectos humanos y de trabajo en equipo (Letelier y Penadés, 2006). Desarrollamos los procesos para XP tablas 3.6 a 3.13 y tablas 3.14 a 3.16 para ICONIX, que han producido software que supera los niveles requeridos de calidad, el estudio llena el vacío descrito por los autores indicados.

CAPITULO V
CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

1. Los requerimientos de calidad del usuario (actor directo) que comercializa tara en la Región Ayacucho y, de los proveedores de bienes y servicios (actores indirectos), han determinado las características de calidad del software de acuerdo al modelo según la NTP-ISO/IEC 9126-1 anexo B y de las sub características relacionadas, medidas y evaluadas usando la NTP-ISO/IEC-TR 9126-3 y la NTP-ISO/IEC-TR 9126-2 anexo C, los atributos de la calidad interna, calidad externa del producto software evaluados se muestran en anexo D, la NTP-ISO 14598-3 se uso para asegurar la calidad del producto software. El control y aseguramiento de calidad de los productos software desarrollados con ICONIX y XP, se realizó de acuerdo a normas aceptadas y estandarizadas en el Perú.
2. Aplicando el marco teórico de XP, construimos el proceso para desarrollar software, donde: la fase de exploración con las tareas, escribir historias de usuario, probar las tecnologías ha utilizar, estimar esfuerzo para las historias de usuario y sus entregables. La fase de planificación las tareas, rescribir las historias de usuario, formular el plan de versiones y sus entregables. La fase de iteración con las tareas, definir la arquitectura técnica, escribir tareas de ingeniería, formular el plan de iteraciones, crear pruebas de aceptación, implementar las interfaces, escribir tarjetas CRC para cada tarea de ingeniería, implementar la base de datos física, implementar código para clases entidad, crear pruebas unitarias para las clases control, implementar código fuente, ejecutar pruebas unitarias, realizar integración continua, ejecutar pruebas de integración para una historia de usuario, ejecutar pruebas de aceptación, y sus entregables. El proceso se describe en las tablas

3.14, 3.15 y 3.16, la fase de mantenimiento esta fuera de los objetivos del estudio. El proceso XP se aplicó para desarrollar el software, los resultados se observan en las tablas 4.9 a 4.46, figuras 4.22 a 4.26 y el anexo A.

De acuerdo al marco teórico para ICONIX, elaboramos el proceso, donde: la fase de análisis de requisitos consta de las tareas, identificar requisitos, identificar objetos del mundo real y dibujar modelo de dominio, realizar prototipo de interfaz gráfica, descubrir casos de uso, dibujar y empaquetar casos de uso, asignar requisitos funcionales a los casos de uso, escribir el primer borrador de casos de uso y, sus entregables. El primer hito revisión de requisitos, revisa el modelo de dominio, prototipo GUI, primer borrador de casos de uso. La fase de diseño preliminar y sus tareas, reescribir el primer borrador para cada caso de uso, identificar el primer corte de objetos que completan escenarios para cada caso de uso, actualizar el modelo de dominio, actualizar el diagrama de clases de análisis y sus entregables. El segundo hito revisión de diseño preliminar, revisa la descripción de casos de uso, diagrama de robustez y modelo de dominio actualizado. La arquitectura técnica no es una fase del proceso ICONIX, pero considera las tareas, diseñar diagrama de componentes, diseñar diagrama de despliegue y sus entregables. La fase de diseño y sus tareas, dividir modelo de dominio actualizado para cada caso de uso, dibujar un diagrama de secuencia para cada caso de uso, actualizar el diagrama de clases de un caso de uso, extraer controladores para pruebas unitarias y sus entregables. El tercer hito revisión crítica de diseño, revisa el diagrama de secuencia, modelo de dominio actualizado, diagrama de clase y lista de pruebas unitarias. La fase de implementación y sus tareas, implementar la base de datos física, implementar código para clases entidad, implementar código para las interfaces, crear pruebas unitarias para cada controlador, implementar código fuente para cada controlador, ejecutar pruebas unitarias para cada controlador, ejecutar pruebas de aceptación para cada caso de uso y sus entregables. El proceso ICONIX esta descrito en las tablas 3.6 a 3.13. El proceso ICONIX se uso para desarrollar el software cuyos

resultados están en las tablas 4.1 a 4.8, figuras 4.1 a 4.21 y el anexo A.

3. Evaluando la calidad interna para XP y ICONIX, no se establece mayor diferencia significativa de sub características y características, tablas 4.58, 4.60, 4.61 y 4.63. La calidad interna del software desarrollado con ICONIX y XP son estadísticamente iguales.

   La evaluación de la calidad externa para XP y ICONIX, muestra que existe mayor diferencia significativa para sub características y características con ICONIX que con XP, ver las tablas 4.64, 4. 66, 4.67 y 4.69. El proceso ICONIX produce para la calidad externa mejor software que el proceso XP.

   La evaluación de la calidad del producto software para comercializar tara en la Región Ayacucho, según la tabla 4.76, para las características y sub-características asociadas a la calidad del producto de ICONIX y XP, según la evidencia estadística indica que no existe diferencia significativa entre la calidad del producto desarrollado con ICONIX y XP.

4. Según el enfoque juicio de expertos, la calidad del producto software para comercializar tara en la Región Ayacucho, aplicando ICONIX, presenta grado de calidad "excede la exigencia", ver tablas 4.53 y 4.54., usando XP, muestra grado de calidad "satisfactorio", ver tablas 4.56 y 4.57.

   Según el análisis estadístico realizado para evaluar la calidad del producto software, con ICONIX, el nivel obtenido (74.07%) es estadísticamente superior al nivel requerido (66.21%), ver tablas 4.71 y 4.72, que indica que el software es de calidad. Aplicando XP, el nivel obtenido es (71.98%) estadísticamente superior al nivel requerido (66.21%), ver tablas 4.74 y 4.75, que también indica software de calidad.

5.2 RECOMENDACIONES

1. Las nuevas líneas de investigación sugeridas a partir del presente estudio son, evaluar la calidad del proceso y del producto para aplicaciones medianas, usando la metodología ágil y formal ICONIX, con la finalidad de tener un análisis cuantitativo de la calidad del producto y del proceso.
2. Estudiar el proceso de enseñanza - aprendizaje de la metodología ágil y

formal ICONIX, a nivel de pre grado, introduciendo mejoras al tratamiento de requisitos, pruebas y disminuir la carga de las diversas revisiones por consumir excesivo tiempo en el modelado y desarrollo.

3. Desarrollar un software genérico aplicando la metodología ICONIX, para la comercialización de cualquier cadena productiva, adicionado la funcionalidad de ventas en línea y, priorizando la característica seguridad de las transacciones comerciales.
4. Estudiar el proceso de enseñanza - aprendizaje de la metodología ágil XP, a nivel de pre grado, introduciendo patrones de diseño estandarizados para la generación de consultas y reportes personalizables de información operativa.
5. Investigar en las empresas desarrolladoras que realizan control y aseguramiento de la calidad de productos software, los pesos y niveles requeridos para las características y sub características del modelo de calidad según la NTP-ISO/IEC 9126-1, a fin de contar con valores más usuales para los software de gestión de negocios.
6. Evaluar la calidad del proceso utilizando la metodología XP, para software pequeño y mediano, que permitiría tener una evidencia cuantitativa de las bondades del proceso XP.

BIBLIOGRAFIA

1. Avendaño, E. et al., (2006) *Análisis Participativo de la Cadena Productiva de Tara en Ayacucho*. Primera Edición. Perú, Ayacucho, IDESI-SNV.
2. Avendaño, E. et al., (2007) *Conociendo la Cadena Productiva de Tara en Ayacucho*. Primera Edición. Perú, Ayacucho, Solid Perú.
3. Ariel, J. y C. Bastarrica, (2005) *Sistema Integral de Mejoramiento de los Procesos de Desarrollo de Software en Colombia*. Investigación. Chile, Departamento de Ciencias de la Computación. Universidad de Chile.
4. Bautista, E. y N. Bendezú, (1998) *Proyecto de Factibilidad para la Utilización Integral del Fruto de la Tara*. Perú, Facultad de Ingeniería Química y Metalurgia. Universidad Nacional de San Cristóbal de Huamanga.
5. Beck, K. (1999) *Extreme Programming Explained: Embrace Change*. Primera Edición. Reading, Massachusetts, Addison-Wesley.
6. Beck, K. y M. Fowler, (2000) *Planning Extreme Programming*. Primera Edición. United States. Addison Weley.
7. Bedini, A. (2000) Calidad Tradicional y de Software. Primera Edición. Chile, Departamento de Industrias. Universidad Técnica Federico Santa María.
8. Bernard, O. (1992). *Estadística Aplicada*. Décima Edición. México. Limusa.
9. Caballero, E. (2007*) Mejora de la Calidad del Software en el Entorno de Microempresas de TI*. Tesis Doctorado. España, Facultad de Informática. Universidad Politécnica de Madrid.
10. Calero, M. (2003) "*Una explicación de la programación extrema (XP)*". [En línea]. España, disponible en: http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/explicaxp.pdf. [Acceso 5 diciembre 2008].
11. Calero, M. (2005) Métricas del Software: Conceptos Básicos, Definición y Formalización. [En línea]. España. Disponible en: http://alarcos.inf-cr.uclm.es/doc/calidadSI/Metodo%20y%20Formalizaci%C3%B3n.ppt. [Acceso 15 marzo 2009].

12. Crosby, P. (1979) *Quality is Free: the Art of Making Quality Certain*. EUA. McGraw Hill.

13. Díaz, J. (2001) *Lineamientos para la Determinación del Perfil de Calidad de un Producto de Software*. Tesis de Maestría. México, Programas de Posgrado en Electrónica, Computación, Información y Comunicaciones. Instituto Tecnológico y de Estudios Superiores de Monterrey.

14. Díaz, J. (2001) *Lineamientos para la Determinación del Perfil de Calidad de un Producto de Software*. Tesis de Maestría. México, Programas de Posgrado en Electrónica, Computación, Información y Comunicaciones. Instituto Tecnológico y de Estudios Superiores de Monterrey.

15. Diez, E. (2003) *Aseguramiento de La Calidad en La Construcción de Sistemas Basados en El Conocimiento un Enfoque Práctico*. Trabajo de Especialización. Argentina, Ingeniería Informática. Instituto Tecnológico de Buenos Aires.

16. De la Cruz, P. (2004) *Aprovechamiento Integral y Racional de la Tara.* Perú, Revista del Instituto de Investigación. Facultad de Ingeniería Geología, Minera, Metalurgia y Geografía. UNMSM.

17. Esparza, P. (2006) *Modelos de Calidad Web: Clasificación de Métricas.* Tesis Licenciatura. España, Escuela Técnica Superior de Ingeniería Informática. Universidad Nacional de Educación a Distancia.

18. Figueroa, R.; Solis, C. y A. Cabrera, (2007) *Metodologías Tradicionales vs. Metodologías Ágiles.* Investigación. Ecuador, Escuela de Ciencias en Computación. Universidad Técnica Particular de Loja.

19. Fowler, M. (2005) *The New Methodology*. [En línea]. EUA. Disponible en: <http://www.programacionextrema.org/articulos/newMethodology.es.html>. [Acceso 12 mayo 2009].

20. Fowler, M. y M. Foemmel (2006) *Continuous Integration*. [En línea]. EUA. Disponible: <http://martinfowler.com/articles/continuousIntegration.html>. [Acceso 12 mayo 2009].

21. González, S. y L. Fernández, (2006) *Programación Extrema: Prácticas, Aceptación y Controversia. En Revista CULCYT* No 14-15, Año 3. Mayo–Agosto 2006. México, Universidad Autónoma de Ciudad Juárez.

22. Hernández, R.; Fernández, C. y P. Baptista (1997) *Metodología de la*

Investigación. Primera Edición. México. McGraw-Hill.

23. Ishikawa, K. (1986) *¿Qué es Control Total de Calidad?: La Modalidad Japonesa*. Barcelona. Editorial Norma.
24. ISO/IEC 9126:1 (1991) *Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their use*. Ginebra. Secretaría General de ISO.
25. Jacobson, I.; Booch, G. y J. Rumbaugh, (2000) *El Proceso Unificado de Desarrollo de Software*. Primera Edición. Madrid, Pearson Educación S.A.
26. Jeffries, R. (2001) *What is Extreme Programming?*. [En línea]. EUA. Disponible: http://www.xprogramming.com/xpmag/. [Acceso 11 mayo 2009].
27. Johnson, R. et al. (2008) Spring java/j2ee Application Framework.
28. Kitzberger, S. (2005) *Framework Iconix: Uma Alternativa para Sistemas de Médio Porte*. Tesis Bachiller. Brasil, Universidade Tecnológica Federal do Paraná.
29. Ladd, S. et al. (2006) *Expert Spring MVC and Web Flow*. Primera Edición. California. Apress.
30. Larman, G. (1999) *UML y Patrones: Introducción al Análisis y Diseño Orientado a Objetos*. Primera Edición. Mexico, Prentice-Hall.
31. Letelier, P. y, M. Penadés, (2006) *Metodologías Ágiles para el Desarrollo de Software: eXtreme Programming (XP)*. Investigación. España, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.
32. Martin, M. (2004) *Sistema de Catalogación de Métricas e Indicadores con Potencia de Web Semántica*. Tesis de Magíster. Argentina, Facultad de Informática. Universidad Nacional de La Plata.
33. MINCETUR-USAID (2005). *Plan Estratégico Regional Exportador - Región Ayacucho*. Primera Edición. Ayacucho. Editorial Mincetur.
34. Montgomery, D. (2002). *Diseño y Análisis de Experimentos*. Segunda Edición. México. Limusa - Wiley.
35. NTP-ISO/IEC 9126-1 (2004) *Ingeniería de Software. Calidad del Producto. Parte 1: Modelo de Calidad*. Primera Edición. Lima. INDECOPI.
36. NTP-ISO/IEC 9126-2 (2004) *Ingeniería de Software. Calidad del Producto. Parte 2: Métricas Externas*. Primera Edición. Lima. INDECOPI.

37. NTP-ISO/IEC 9126-3 (2005) *Ingeniería de Software. Calidad del Producto. Parte 3: Métricas Internas*. Primera Edición. Lima. INDECOPI.

38. NTP-ISO/IEC 14598-1 (2005) *Tecnología de la Información. Evaluación del Producto. Parte 1: Visión General.* Primera Edición. Lima. INDECOPI.

39. NTP-ISO 14598-3 (2005) *Ingeniería de Software. Evaluación del Producto. Parte 3: Proceso para Desarrolladores.* Primera Edición. Lima. INDECOPI.

40. Pantoja, J. (2005) "*Construindo Softwares com Qualidade e Rapidez Usando ICONIX*". [En línea]. Brasil, disponible en: http://www.guj.com.br/article.show.logic?id=172. [Acceso 15 octubre 2008].

41. Pillpe, B. y L. Quispe, (2007) *Factores que Limitan la Exportación de Productos Ayacuchanos – Caso: Tara*. Tesis. Perú, Facultad de Ciencias Económicas y Administrativas, Universidad Nacional de San Cristóbal de Huamanga.

42. Pressman, R. (2002) *Ingeniería del Software: Un Enfoque Práctico*. 5ta Edición. España, McGraw-Hill/Interamericana de España.

43. Rafael, M. (2004) *Processos de Desenvolvimento de Software: RUP e ICONIX*. Tesis Bachiller. Brasil. Universidade Estadual de Londrina.

44. Reynoso, C. (2004) *Métodos Heterodoxos en Desarrollo de Software*. Argentina, Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

45. Rosenberg, D.; Stephens, M. y M. Collins-Cope, (2005) *Agile Development with ICONIX Process: People, Process, and Pragmatism*. Primera Edición. EUA, Apress.

46. Rosenberg, D. y K. Scott (1999) *Use Case Driven Object Modeling with UML: A Practical Approach*. Primera Edición. EUA. Addison-Wesley.

47. Rosenberg, D. y K. Scott (2001) *Applyging Use Case Driven Object Modeling with UML: Un Anotated e- Commerce Sample*. Primera Edición. United States. Addison-Wesley.

48. Rosenberg, D. y M. Stephens (2007) *Use Case Driven Object Modeling with UML: Theory and Practice*. Primera Edición. United States. Apress.

49. Rodríguez, D. y R. Harrison (1997) *Medición en la Orientación a Objetos*. [En línea]. UK. En: <http://www.cc.uah.es/drg/b/RodHarRama00.pdf>. [Acceso 25 marzo 2009].

50. Royce, W. (1970) *Managing the Development of Large Software Systems: Concepts and Techniques*. [En línea]. Alemania, disponible en: http://www.informatik.uni-bremen.de/gdpa/def/def_w/WATERFALL.htm. [Acceso 10 abril 2009].

51. Rumpe, B. y A. Schröder, (2001) *Quantitative Survey on Extreme Programming Projects*. Investigación. Alemania, Software & Systems Engineering. Munich University of Technology.

52. Santana, R. (2003) *Calidad en el Desarrollo de Software en el Gobierno del Estado de Tamaulipas y Alternativas de Mejora*. Tesis de Maestría. México, Unidad Académica Multidisciplinaría de Comercio. Universidad Autónoma de Tamaulipas.

53. Sánchez, M.; Letelier, P. y J. Canós, (2000) *Código de Calidad: Integrando Patrones de Diseño y Refactoring*. Investigación. España, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.

54. Salazar, M. y D. Van Der Heyden (2004) *Metodología de Análisis de Cadenas Productivas con Equidad para la Promoción del Desarrollo Local*. Primera Edición. Lima. Editorial SNV.

55. Scalone, F. (2006) *Estudio Comparativo de Los Modelos y Estándares de Calidad del Software*. Tesis de Magíster. Argentina, Maestría en Ingeniería en Calidad. Universidad Tecnológica Nacional.

56. Schach, S. (2005) *Análisis y Diseño Orientado a Objetos con UML y El Proceso Unificado*. Primera Edición. United States, McGraw-Hill.

57. Sicilia, M.; Ortega, J. y E. García, (2002) *Integración de Procedimientos de Control en Prácticas de Desarrollo XP a través del Concepto de Micro*-Práctica. Investigación. España, Departamento de Informática. Universidad Carlos III.

58. Soares, M. (2009) "*Processo de Modelagem de Software ICONIX*". [En línea]. Brasil, disponible en: ftp://www.autocominformatica.com.br/pub/ProcessoICONIX.pdf. [Acceso 25 abril 2009].

59. Sommerville, I. (2005) *Ingeniería del Software*. Séptima Edición. España. Madrid, Pearson Educación.

60. Stephens, M. y D. Rosenberg (2003) *Extreme Programming Refactored:*

The Case Against XP. Primera Edición. California. Apress.

61. Van Der Heyden, D. y P. Camacho (2006) *Guía Metodológica para el Análisis de Cadenas Productivas*. Segunda Edición. Quito. Editorial Plataforma RURALTER.

62. Wake, W. (2002) *XP123 - Exploring Extreme Programming*. [En línea]. EUA. Disponible en: <http://www.xp123.com/>. [Acceso 13 mayo 2009].

63. Wells, D. (2006) *Extreme Programming: A Gentle Introduction*. [En línea]. EUA. Disponible: <http://www.extremeprogramming.org/index.html>. [Acceso 12 mayo 2009].

ANEXO A

ARTEFACTOS DEL SOFTWARE PARA COMERCIALIZAR TARA DE LA REGION AYACUCHO

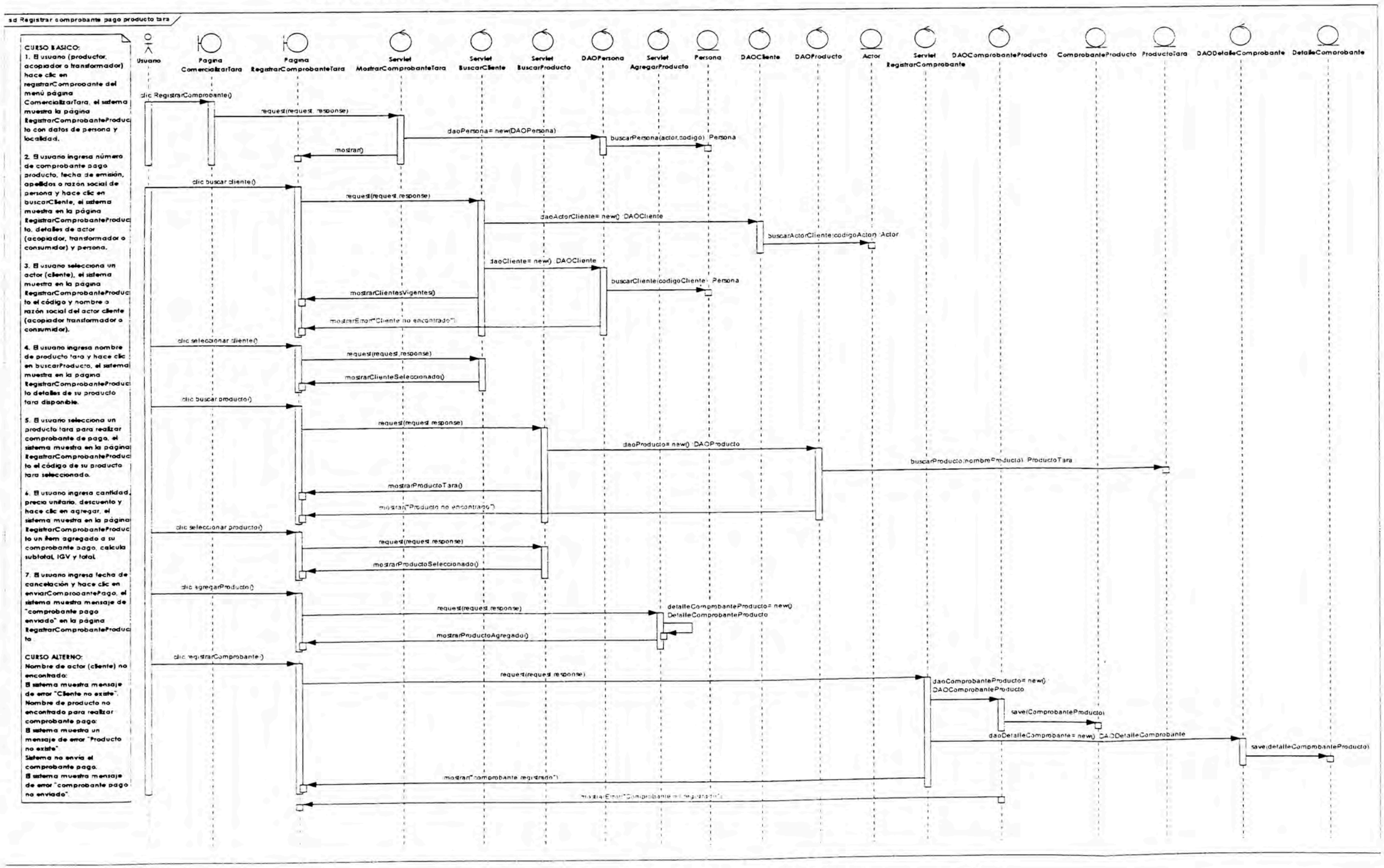

Figura N° A.1: Diagrama de secuencia. Registrar comprobante pago producto tara - ICONIX

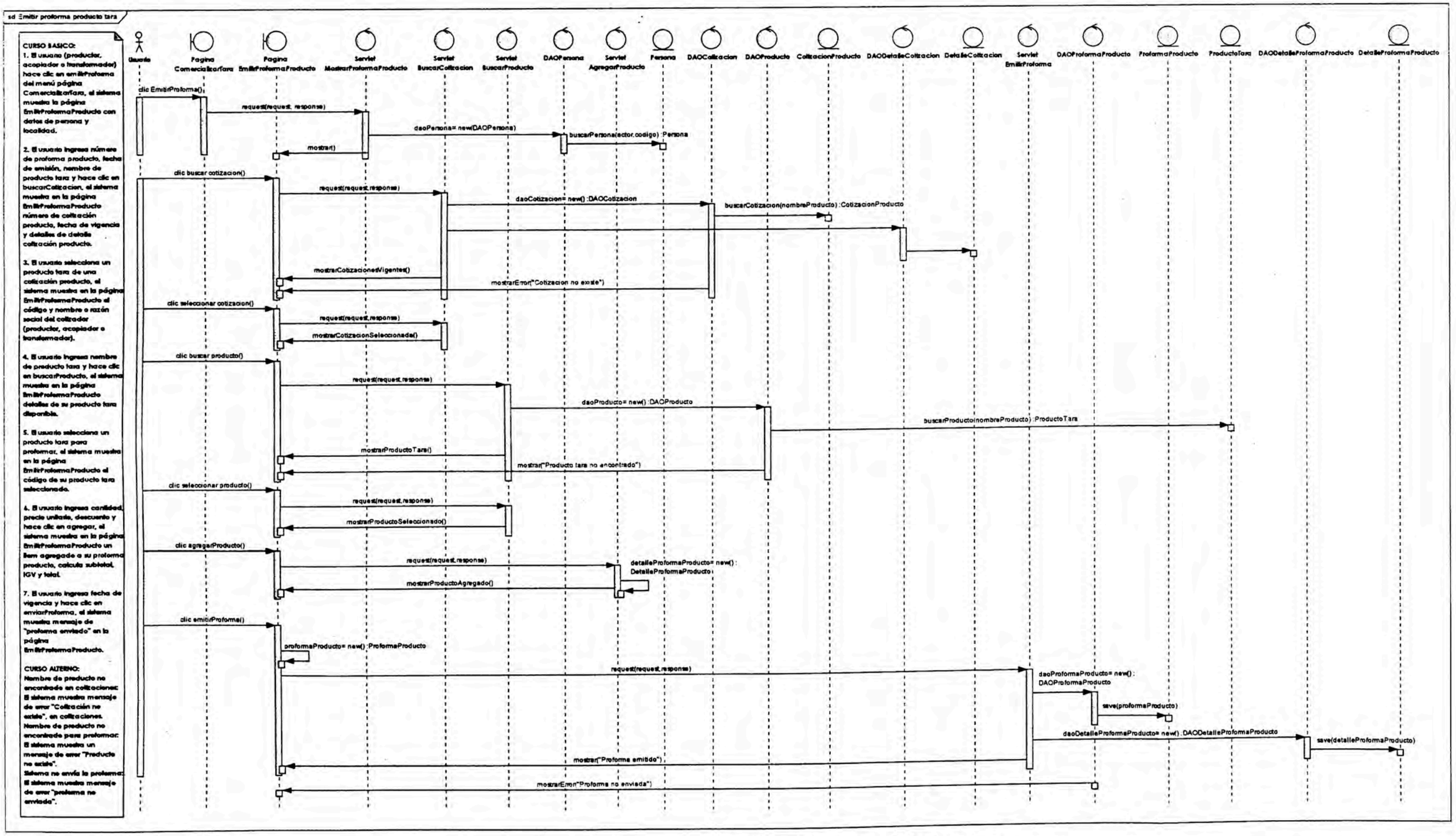

Figura N° A.2: Diagrama de secuencia. Emitir proforma producto tara - ICONIX

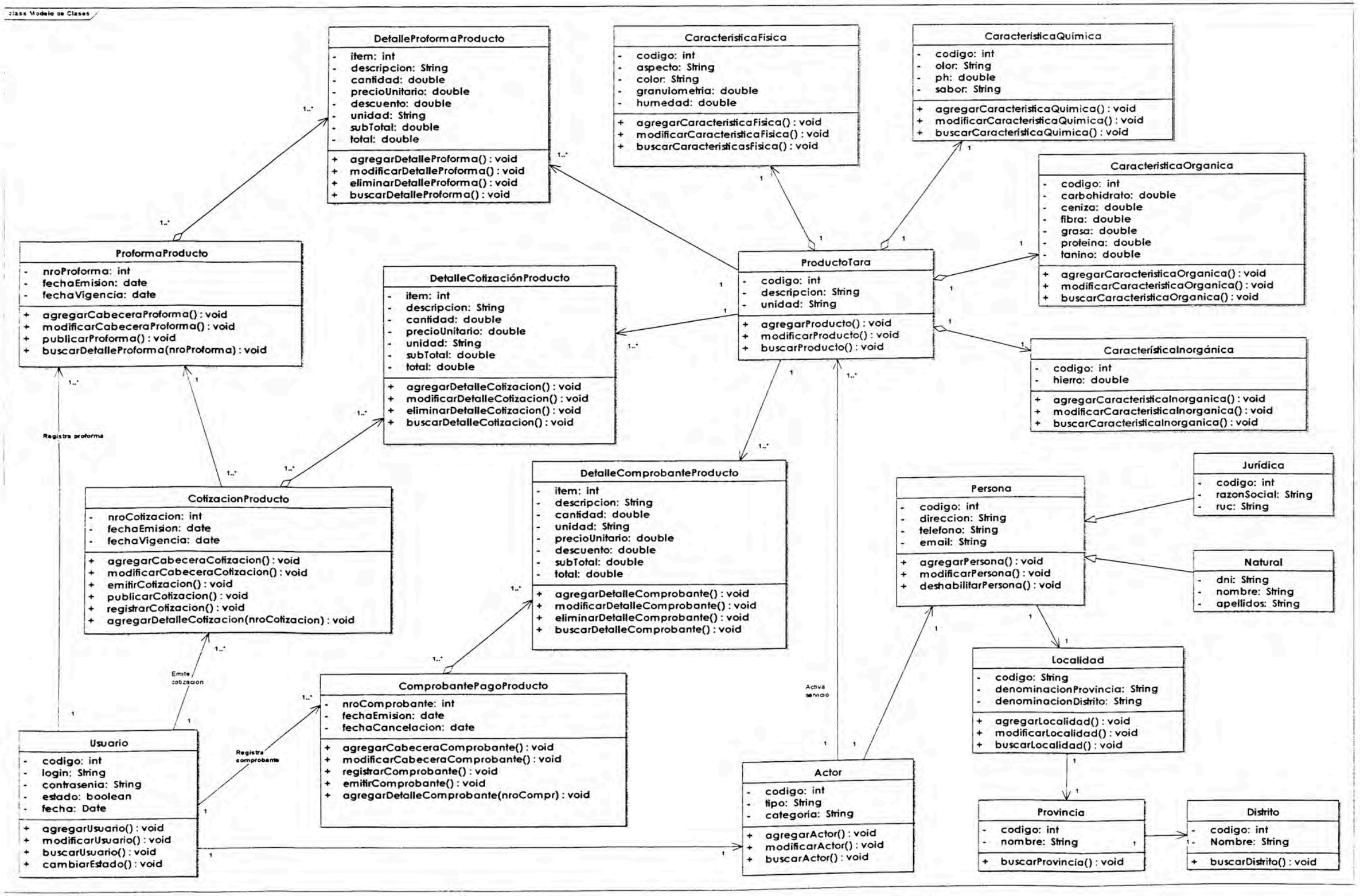

Figura N° A.3: Diagrama de clases actualizado para dos casos de uso - ICONIX

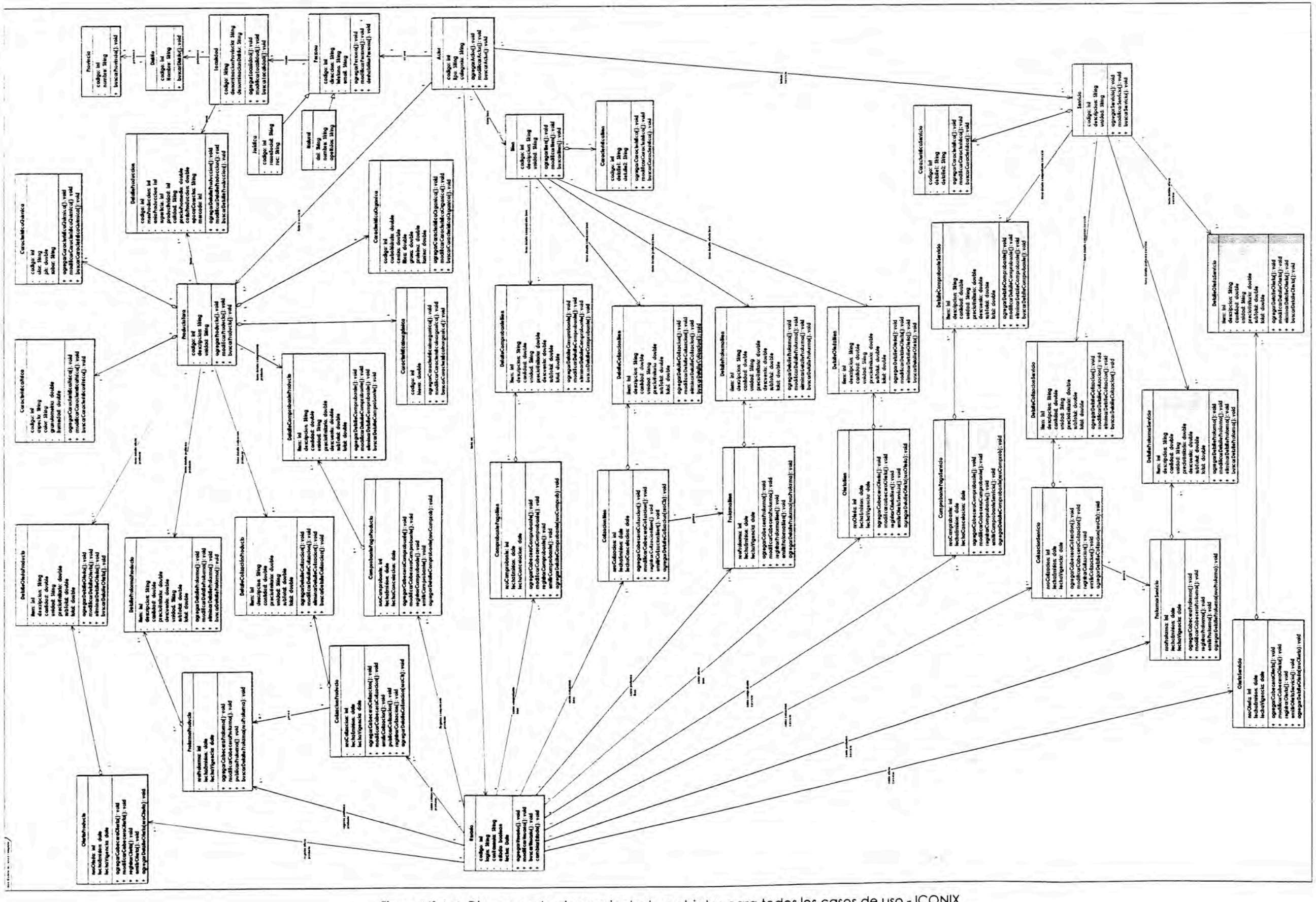

Figura N° A.4: Diagrama de clases orientado a objetos para todos los casos de uso - ICONIX

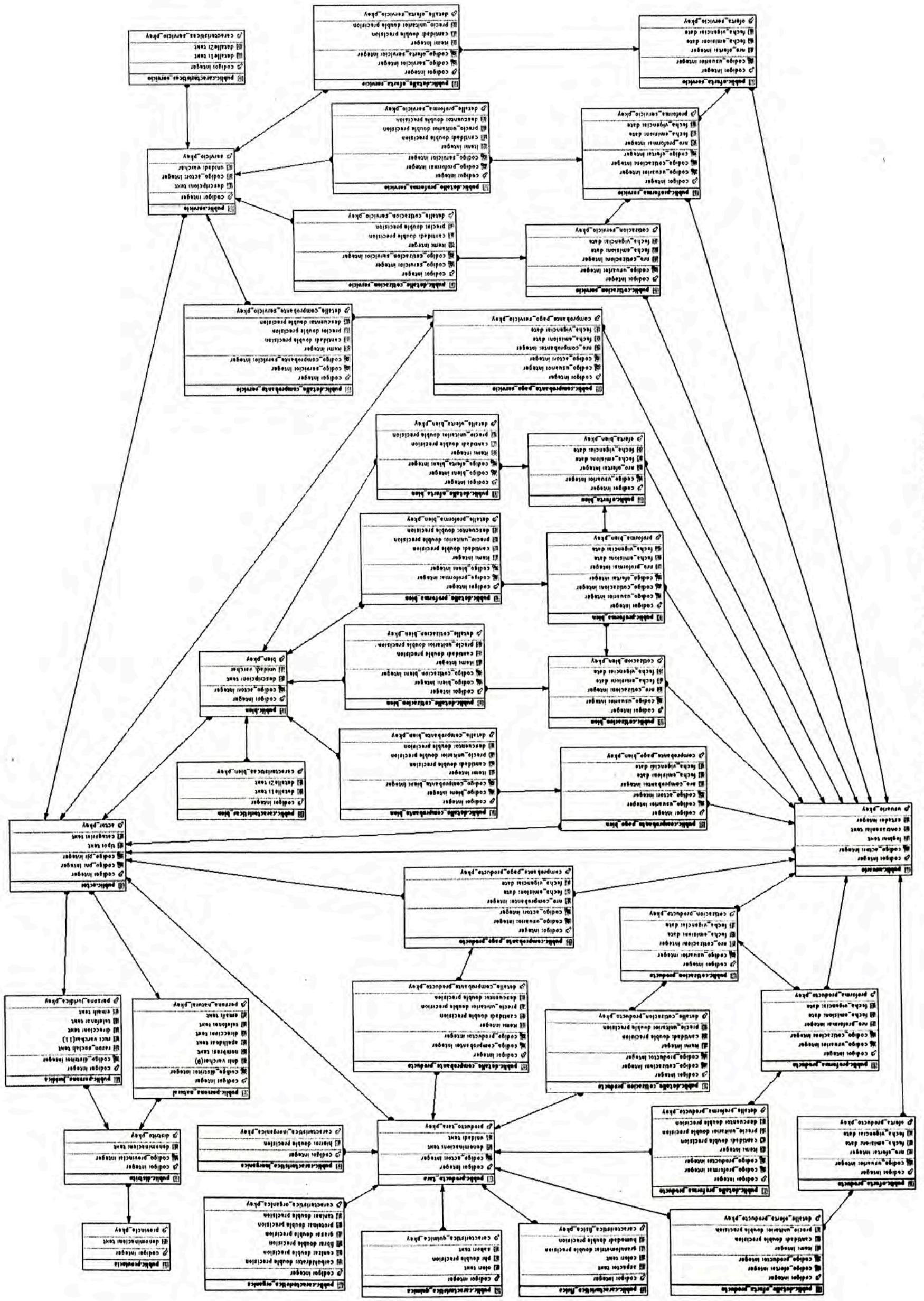

Figura N° A.5: Base de datos para todos los casos de uso - ICONIX

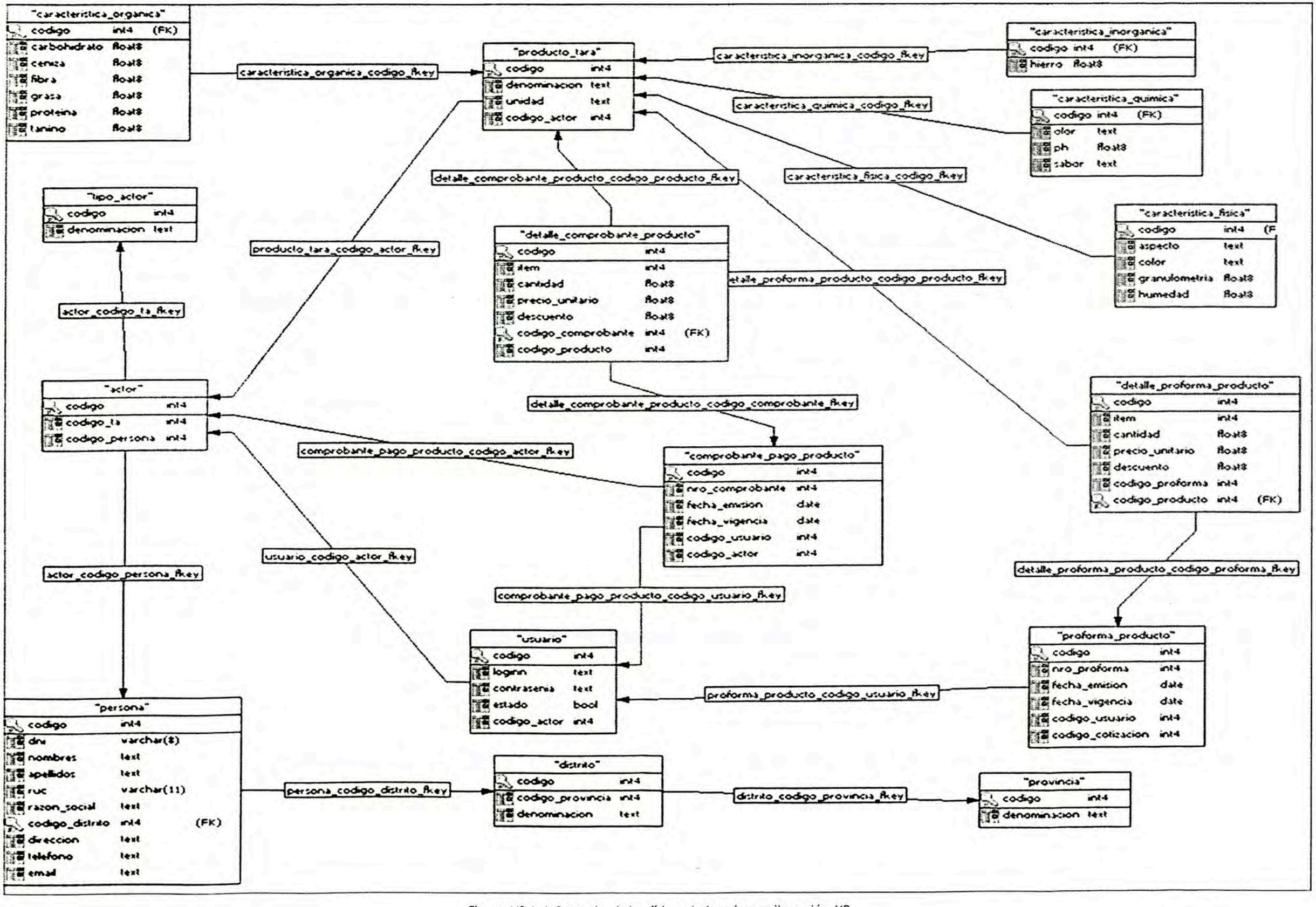

Figura N° A.6: Base de datos física de la primera iteración-XP

ANEXO B
CARACTERÍSTICAS Y SUB CARACTERÍSTICAS DEL MODELO DE CALIDAD (NTP-ISO/IEC 9126)

B.1 FUNCIONALIDAD

La capacidad del producto software para proveer las funciones que satisfacen las necesidades explícitas e implícitas cuando el software se utiliza bajo condiciones específicas.

Aplicabilidad.- La capacidad del producto software para proveer un conjunto de funciones para las tareas y objetivos especificados por el usuario.

Precisión.- La capacidad del producto software para proveer los resultados o efectos acordados con un grado necesario de precisión.

Interoperabilidad.- La capacidad del producto software a interactuar con uno o más sistemas especificados.

Seguridad.- La capacidad del producto software para proteger la información y los datos de modo que las personas o los sistemas no autorizados no puedan leerlos o modificarlos y a las personas o sistemas autorizados no se les denegará el acceso.

Conformidad de la funcionalidad.- La capacidad del producto software de adherirse a los estándares, convenciones o regulaciones legales y prescripciones similares referentes a la funcionalidad.

B.2 FIABILIDAD

La capacidad del producto software para mantener un nivel específico de funcionamiento cuando se está utilizando bajo condiciones especificadas.

Madurez.-La capacidad del producto software para evitar fallos como resultado de defectos en el software.

Tolerancia a fallas.- La capacidad del producto software para mantener un nivel especificado de funcionamiento en caso de defectos del software o de incumplimiento de su interfaz especificada.

Recuperabilidad.- La capacidad del producto software para reestablecer un nivel especificado de funcionamiento y recuperar los datos afectados directamente en el caso de un fallo.

Conformidad de la fiabilidad.- La capacidad del producto software para

adherirse a las normas, convenciones o regulaciones relativas a la fiabilidad.

B.3 USABILIDAD

La capacidad del producto software de ser entendido, aprendido, usado y atractivo al usuario, cuando es usado bajo las condiciones especificadas.

Entendibilidad.- La capacidad del producto software para permitir al usuario entender si el software es aplicable, y cómo puede ser utilizado para las tareas y las condiciones particulares de la aplicación.

Facilidad de aprendizaje.- La capacidad del producto software para permitir al usuario aprender su aplicación.

Operabilidad.- La capacidad del producto software para permitir al usuario operarlo y controlarlo.

Atractividad.- La capacidad del producto software de ser atractivo al usuario.

Conformidad de usabilidad.-La capacidad del producto software para adherirse a los estándares, convenciones, guías de estilo o regulaciones relacionadas a su usabilidad.

B.4 EFICIENCIA

La capacidad del producto software para proveer un desempeño apropiado, de acuerdo a la cantidad de recursos utilizados y bajo las condiciones planteadas.

Comportamiento en el tiempo.- La capacidad del producto software para proveer tiempos apropiados de respuesta y procesamiento, y ratios de rendimiento cuando realiza su función bajo las condiciones establecidas.

Utilización de recursos.- La capacidad del producto software para utilizar apropiadas cantidades y tipos de recursos cuando este funciona bajo las condiciones establecidas.

Conformidad de eficiencia.- La capacidad del producto software para adherirse a normas o convenciones relacionadas a la eficiencia.

B.5 FACILIDAD DE MANTENIMIENTO

Capacidad del producto software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, y en requerimientos y especificaciones

funcionales.

Analizabilidad.- La capacidad del producto software para ser diagnosticado por deficiencias o causas de fallos en el software o la identificación de las partes a ser modificadas.

Cambiabilidad.- La capacidad del software para permitir que una determinada modificación sea implementada.

Estabilidad.- La capacidad del producto software para evitar efectos inesperados debido a modificaciones del software.

Testeabilidad.- La capacidad del software para permitir que las modificaciones puedan ser validadas.

Conformidad de facilidad de mantenimiento.- La capacidad del software para adherirse a estándares o convenciones relativas a la facilidad de mantenimiento.

B.6 PORTABILIDAD

La capacidad del software para ser trasladado de un entorno a otro.

Adaptabilidad.- La capacidad del producto software para ser adaptado a diferentes entornos definidos sin aplicar acciones o medios diferentes de los previstos para el propósito del software considerado.

Instalabilidad.- La capacidad del producto software para ser instalado en un entorno definido.

Co existencia.- La capacidad del producto software para coexistir con otro producto software independiente dentro de un mismo entorno compartiendo recursos comunes.

Reemplazabilidad.- La capacidad del producto software para ser utilizado en lugar de otro producto software, para el mismo propósito y en el mismo entorno.

Conformidad de portabilidad.- La capacidad del software para adherirse a estándares o convenciones relacionados a la portabilidad.

ANEXO C

METRICAS UTILIZADAS

METRICAS INTERNAS

C.1 METRICAS DE FUNCIONALIDAD

C.1.1 METRICAS DE APLICABILIDAD

Nombre	**adecuación funcional**
Propósito de la métrica	¿Cuán adecuadas son las funciones revisadas?
Método de aplicación	Contar el número de funciones implementadas en las que se detectó problemas para realizar las tareas especificadas y comparar con las funciones implementadas. Se puede medir lo siguiente: - todas o partes de las especificaciones de diseño - módulos/partes completadas de productos software
Medición, fórmula	X = 1-A/B A = número de funciones en las que se detecto problemas durante la evaluación B = numero de funciones revisadas
Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Código fuente Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.1: Métrica de aplicabilidad - adecuación funcional

Nombre	**integridad de implementación funcional**
Propósito de la métrica	¿Cuán completa esta la implementación funcional?
Método de aplicación	Contar el número de funciones faltantes detectadas en la evaluación y comparar con el número de funciones descritas en la especificación de requerimientos
Medición, fórmula	X = 1-A/B A = Número de funciones faltantes detectadas en evaluación B = Número de funciones descritas en la especificación de requerimientos
Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1 es lo mejor

Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Código fuente Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.2: Métrica de aplicabilidad - integridad de implementación funcional

Nombre	**cobertura de la implementación funcional**
Propósito de la métrica	¿Cuán correcta es la implementación funcional?
Método de aplicación	Contar el número de funciones faltantes o implementaciones incorrectamente y comparar con el número de funciones descritas en la especificación de requerimientos
Medición, fórmula	X = 1-A/B A = Número de funciones faltantes o implementaciones incorrectamente que se detectaron B = Número de funciones descritas en la especificación requerimientos
Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Código fuente Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.3: Métrica de aplicabilidad – cobertura de implementación funcional

C.1.2 METRICAS DE PRECISION

Nombre	**exactitud de cálculos**
Propósito de la métrica	¿Cuán completamente se implementaron los requerimientos de exactitud?
Método de aplicación	Contar el número de funciones que han implementado los requerimientos de exactitud y comparar con el número de funciones con requerimientos de exactitud especificaciones
Medición,	X = A/B

fórmula	A = Número de funciones en la que se han implementado requerimientos de exactitud específicos, confirmados en la evaluación B = Número de funciones para las cuales se necesita implementar requerimientos de exactitud específicos
Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de Requerimientos Diseño Código fuente Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.4: Métrica de precisión - exactitud de cálculo

C.2 METRICAS DE FIABILIDAD

C.2.1 METRICAS DE MADUREZ

Nombre	**eliminación de fallas**
Propósito de la métrica	¿Cuántas fallas fueron corregidas? ¿Cuál es la proporción de fallas removidas?
Método de aplicación	Contar el número de fallas corregidas durante el diseño/codificación. Contar el número de fallas removidas durante el diseño/codificación y comparar con el número de fallas detectadas en la revisión durante el diseño/codificación.
Medición, fórmula	X = A A = número de fallas corregidas en diseño/codificación. Y = A/B A = número de fallas corregidas en diseño/codificación. B = número de fallas detectadas en las revisión.
Interpretación del valor medido	0 <= X, Un valor alto de X implica quedan menos fallas 0<=Y<=1, Mientras mas cercano a 1 mejor (mas fallas removidas)
Tipo de escala	Ratio, Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	El valor A proviene del reporte de remoción de fallas El valor B proviene del reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.6 Revisión conjunta
Audiencia	Evaluador

objetivo	Desarrollador

Tabla C.5: Métrica de madurez – eliminación de fallas

C.2.2 METRICAS DE TOLERANCIA A FALLOS

Nombre	**prevención de operación incorrecta**
Propósito de la métrica	¿Cuántos funciones se han implementado con capacidad de prevención de operación incorrectas?
Método de aplicación	Contar el número de funciones implementadas para evitar fallas críticas y serias causadas por operación incorrecta y comparar con el número de patrones de operación incorrecta que deben considerarse **Comentario**: Las fallas del sistemas incluyen también datos dañados
Medición, fórmula	X = A/B A = Número de funciones implementadas para evitar patrones de operación incorrecta. B = Número de patrones de operación incorrecta que deben considerarse. **Comentarios.-** Patrones de operación incorrecta son; tipos de datos incorrectos como parámetros, secuencia de datos de entrada incorrecta, secuencia de operación incorrecta. **Comentario.-** La técnica del análisis del árbol de fallas se puede usar para detectar patrones de operación incorrecta.
Interpretación del valor medido	0 <= X , Cuando "X" es mayor mejor es la prevención de operación incorrecta
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	El valor A proviene del reporte de revisión. El valor B proviene del documento de especificación de requisitos.
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.5 Validación 6.6 Revisión conjunta 6.8 Resolución de problemas
Audiencia objetivo	Evaluador Desarrollador Responsable de mantenimiento

Tabla C.6: Métrica tolerancia a fallos - prevención de operación incorrecta

C.3 METRICAS DE USABILIDAD

C.3.1 METRICA DE ENTENDIBILIDAD

Nombre	**función de comprensión**
Propósito de la métrica	¿Qué proporción de las funciones del producto será el usuario capaz de entender en forma correcta?
Método de aplicación	Contar el número de funciones presentes en las interfaces donde el propósito es entendible y comparar con el número de funciones presentes en la interfaz de los usuarios
Medición, fórmula	X = A/B A = Número de funciones presentes en las interfaces de los usuarios cuyo propósito es entendido por el usuario.

	B = Número total de funciones presentes en las interfaces de los usuarios.
Interpretación del valor medido	0 <= X <=1, Lo más cerca de 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.5 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.7: Métrica de entendibilidad - función de comprensión

C.3.2 METRICA DE FACILIDAD DE APRENDIZAJE

Nombre	**integridad de la documentación del usuario y/o facilidad de ayuda**
Propósito de la métrica	¿Qué proporción de las funciones son descritas en la documentación para el usuario y/o facilidades de ayuda?
Método de aplicación	Contar el número de funciones implementadas con facilidades de ayuda y/o documentación y comparar con el número total de funciones del producto
Medición, fórmula	X = A/B A = Número de funciones descritas. B = Número total de funciones proveídas.
Interpretación del valor medido	0 <= X <=1, Lo cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.8: Métrica facilidad de aprendizaje - integridad de la documentación del usuario y/o facilidad de ayuda

C.3.3 METRICA DE OPERABILIDAD

Nombre	**claridad de mensajes**
Propósito de la métrica	¿Qué proporción de los mensajes son auto-explicativos?

Método de aplicación	Contar el número de mensajes implementados con explicaciones claras y comparar con el número de mensajes
Medición, fórmula	X = A/B A = Número de mensajes implementados con explicaciones claras B = Número total de mensajes implementados
Interpretación del valor medido	0 <= X <=1, Lo mas cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia objetivo	Evaluador Desarrollador

Tabla C.9: Métrica de operabilidad - claridad de mensajes

Nombre	**claridad de la interfaz**
Propósito de la métrica	¿Qué proporción de los elementos de la interfaz son auto-explicativos?
Método de aplicación	Contar el número de elementos de la interfaz que sean auto explicativos y comparar con el número total de elementos de la interfaz.
Medición, fórmula	X = A/B A = Número de elementos de interfaz que son auto explicativos B = Número total de elementos de interfaz
Interpretación del valor medido	0 <= X <=1, Lo mas cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia objetivo	Desarrollador Evaluador

Tabla C.10: Métrica de operabilidad - claridad de la interfaz

C.4 METRICA DE EFICIENCIA

C.4.1 METRICA DE UTILIZACION DE RECURSOS

Nombre	**utilización de memoria**
Propósito de	¿Cuál es el tamaño estimado de memoria que ocupará el

la métrica	producto para completar una tarea específica?
Método de aplicación	Estimar el requerimiento de memoria
Medición, fórmula	X = Tamaño en bytes (calculado o simulado)
Interpretación del valor medido	Lo menor es lo mejor
Tipo de escala	Ratio
Tipo de medida	X = Tamaño
Entrada para la medición	Estimar el tamaño de utilización de memoria
Referencia PCVS ISO/IEC 12207	6.4 Verificación
Audiencia objetivo	Desarrollador

Tabla C.11: Métrica de utilización de recursos - utilización de memoria

C.5 METRICA DE FACILIDAD DE MANTENIMIENTO

C.5.1 METRICA DE ANALIZABILIDAD

Nombre	**preparación de funciones de diagnóstico**
Propósito de la métrica	¿Qué tan completa es la previsión de funciones de diagnóstico?
Método de aplicación	Contar el número de funciones de diagnostico implementadas como se han especificado y comparar con el número de funciones de diagnóstico requeridas en la especificación. **Comentario:** esta métrica también es usada para medir la capacidad de análisis de fallas y la capacidad de análisis de causas
Medición, fórmula	X = A/B A = Número de funciones de diagnóstico especificadas implementadas y, confirmadas en la revisión. B = Número de funciones de diagnóstico requeridas.
Interpretación del valor medido	0 <= X <= 1 Lo más cerca a 1, provee una mejor implementación de las funciones de diagnóstico. **Comentario:** Es necesario convertir este valor al intervalo <0,1> si se hace un resumen de las características
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	El valor de A viene del reporte de revisión. El valor de B viene del requerimiento de especificaciones.
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia	Mantenimiento

objetivo	Usuario

Tabla C.12: Métrica de analizabilidad - preparación de funciones de diagnóstico

C.5.2 METRICA DE CAMBIABILIDAD

Nombre	**Registro de cambios**
Propósito de la métrica	¿Son los cambios a las especificaciones y módulos de programa son registrados adecuadamente en el código y haciendo uso de comentarios?
Método de aplicación	Registrar el ratio del módulo de cambio de información
Medición, fórmula	X = A/B A = Número de cambios en funciones y/o módulos que tienen comentarios, confirmado en la revisión. B = Número total de funciones y/o módulos alterados desde la primera versión del código.
Interpretación del valor medido	0 <= X <= 1 Lo mas cerca a 1, indica un mayor registro. Cuando el control de cambio indica 0, significa un pobre control de cambios o pequeños cambios, alta estabilidad.
Tipo de escala	Absoluta.
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Sistema configuración de control Registro de versiones Especificaciones
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia objetivo	Desarrollador Mantenimiento Evaluador

Tabla C.13: Métrica de cambiabilidad - registro de cambios

C.6 METRICAS DE PORTABILIDAD

C.6.1 METRICA DE ADAPTABILIDAD

Nombre	**adaptabilidad al entorno organizacional (adaptabilidad a la organización y a la estructura de la misma)**
Propósito de la métrica	¿Cuán adaptable es el producto al cambio organizacional?
Método de aplicación	Contar el número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en organizaciones múltiples según lo especificado y comparar con el número de funciones con requisitos de adaptabilidad al entorno organizacional.
Medición, fórmula	X = A/B A = Número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en el ambiente de organizaciones y de negocio múltiples según lo especificado, confirmado en la revisión. B = Número total de funciones con requisitos de adaptabilidad al ambiente de la organización.

Interpretación del valor medido	0 <= X <= 1, Lo mas cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Diseño Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.4 Verificación 6.6 Revisión conjunta
Audiencia objetivo	Desarrollador Mantenimiento Evaluador

Tabla C.14: Métrica de adaptabilidad - adaptabilidad al entorno organizacional

C.6.2 METRICA DE INSTALABILIDAD

Nombre	**facilidad de reinstalación**
Propósito de la métrica	¿Cuan fácil es repetir el proceso de reinstalación?
Método de aplicación	Contar el número de reinstalaciones implementadas y comparar con el número de operaciones de reinstalación requeridas.
Medición, fórmula	X = A/B A = Número de reinstalaciones implementadas, confirmadas en revisión. B = Número total de operaciones de instalación requeridas.
Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Reporte de revisión
Referencia PCVS ISO/IEC 12207	6.5 Validación
Audiencia objetivo	Desarrollador

Tabla C.15: Métrica instalabilidad - facilidad de reinstalación

METRICAS EXTERNAS

C.7 METRICAS DE FUNCIONALIDAD

C.7.1 METRICAS DE APLICABILIDAD

Nombre	**adecuación funcional**
Propósito	¿Cuán adecuadas son las funciones evaluadas?
Método de Aplicación	Número de funciones que son adecuados para realizar las tareas especificas comparadas con el número de funciones evaluadas
Medición, fórmula	X = 1-A/B A = Número de funciones en que se detectaron los problemas en la evaluación B = Número de funciones evaluadas
Interpretación del valor medido	0 <= X <= 1, lo mas cerca de 1,0 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Reporte de evaluación
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.3 Aseguramiento de calidad 5.3 Pruebas de calificación
Audiencia objetivo	Desarrollador Responsable de ACS

Tabla C.16: Métrica de aplicabilidad - adecuación funcional

Nombre	**integridad de implementación funcional**
Propósito de la métrica	¿Cuan completa es la implementación de acuerdo a la especificación de requerimientos?
Método de aplicación	Realizar pruebas funcionales (pruebas de caja negra) del sistema de acuerdo a la especificación de requerimientos. Contar el número de funciones faltantes detectadas en la evaluación y compararlas con el número de funciones descritas en la especificación de requerimientos
Medición, fórmula	X = 1-A/B A = Número de funciones faltantes detectadas en la evaluación B = Número de funciones descritas en los especificación de requerimientos
Interpretación del valor medido	0 <= X <= 1, lo mas cerca de 1,0 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Reporte de evaluación
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.3 Aseguramiento de calidad 5.3 Pruebas de calificación
Audiencia	Desarrollador

objetivo	Responsable de ACS

Tabla C.16: Métrica de aplicabilidad - integridad de implementación funcional

Nombre	**cobertura de implementación funcional**
Propósito de la métrica	¿Cuán correcta es la implementación funcional?
Método de aplicación	Realizar pruebas funcionales (pruebas de caja negra) del sistema de acuerdo con la especificación de requerimientos. Contar el número de funciones implementadas incorrectamente o faltantes detectadas en la evaluación y compárelas con el número de funciones descritas en la especificación de requerimientos. Contar el número de funciones que están completas versus las que no lo están
Medición, fórmula	X = 1-A/B A = Número de funciones incorrectamente implementadas o faltantes en la evaluación. B = Número de funciones descritas en la especificación de requerimientos.
Interpretación del valor medido	0 <= X <= 1, Cuanto más se acerca de 1,0 es la mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A = Cantidad B = Cantidad
Entrada para la medición	Especificación de requerimientos Reporte de evaluación
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.3 Aseguramiento de calidad 5.3 Pruebas de calificación
Audiencia objetivo	Desarrollador Responsable de ACS

Tabla C.17: Métrica de aplicabilidad – cobertura de implementación funcional

C.7.2 METRICA DE PRECISION

Nombre	**exactitud de cálculos**
Propósito de la métrica	¿Cuan frecuente los usuarios finales encuentran resultados inexactos?
Método de aplicación	Registrar el número de cálculos inexactos basados en especificaciones
Medición, fórmula	X = A/T A = Número de cálculos inexactos encontrados por los usuarios T = Tiempo de operación
Interpretación del valor medido	0 <= X, lo mas cercano a 0,0 es lo mejor
Tipo de escala	Ratio
Tipo de medida	X = Cantidad/tiempo A = Cantidad T = Tiempo
Entrada para la medición	Especificación de requerimientos Reporte de pruebas
Referencia PCVS ISO/IEC 12207	6.5 Validación 6.3 Aseguramiento de la calidad
Audiencia	Desarrollador

objetivo	Responsable de ACS

Tabla C.18: Métrica de exactitud - exactitud de cálculo

C.8 METRICAS DE FIABILIDAD

C.8.1 METRICAS DE MADUREZ

Nombre	**densidad de fallas contra los casos de prueba**
Propósito de la métrica	¿Cuántas fallas fueron detectados durante el período de prueba definido?
Método de aplicación	Contar el número de fallos detectados y casos de prueba ejecutados
Medición, fórmula	X = A1/A2 A1 = Número de fallas detectadas A2 = Número de casos de prueba ejecutados
Interpretación del valor medido	0 <= X Depende de la fase a probar. En las fases finales, cuanto mas pequeño el valor es mejor
Tipo de escala	Absoluta
Tipo de medida	X = Cantidad/Cantidad A1 = Cantidad A2 = Cantidad
Entrada para la medición	Reporte de prueba Reporte de operación Reporte del problema
Referencia PCVS ISO/IEC 12207	5.3 Integración 5.3 Pruebas de calificación 5.4 Operación 6.3 Aseguramiento de la calidad
Audiencia objetivo	Desarrollador Responsable de pruebas Responsable de ACS

Tabla C.19: Métrica de madurez - densidad de fallas contra los casos de prueba

Nombre	**madurez de la prueba**
Propósito de la métrica	¿Es el producto bien probado? Comentario (s). Esto es para predecir la proporción de éxito que el producto lograra en pruebas futuras
Método de aplicación	Contar el número de casos de prueba pasados que han sido realmente ejecutados y comparar el número total de casos de prueba ha ser ejecutados según las requisitos
Medición, fórmula	X = A/B A = Número de casos de prueba pasados durante la prueba y operación B = Número de casos de prueba ha ser ejecutados para respaldar los requerimientos
Interpretación del valor medido	0 <= X <= 1, El valor más cercano a 1,0 es el mejor
Tipo de escala	Absoluta
Tipo de medida	A = Cantidad B = Cantidad X = Cantidad/Cantidad
Entrada para la medición	Especificación de requerimientos Especificación de pruebas

	Manual de usuario Reporte de pruebas Reporte de operación
Referencia PCVS ISO/IEC 12207	5.3 Pruebas de calificación 6.3 Aseguramiento de la calidad
Audiencia objetivo	Desarrollador Responsable de pruebas Responsable de ACS

Tabla C.20: Métrica de madurez - madurez de la prueba

C.8.2 METRICA DE TOLERANCIA A FALLOS

Nombre	**prevención de operación incorrecta**
Propósito de la métrica	¿Cuántas funciones son implementadas con capacidad de prevención de operación incorrecta?
Método de aplicación	Contar el número de casos de prueba de operaciones incorrectos cuando son previstas para causar fallas críticas y serias y comparar con el número de casos de prueba de tipos de operaciones incorrectas ha ser considerados
Medición, fórmula	X = A/B A = Número de ocurrencias de fallas críticas y serias B = Número de casos de pruebas ejecutados de modelos incorrectos de operación durante la prueba (falla casi causada)
Interpretación del valor medido	0 <= X <= 1 El valor más cercano a 1,0 es lo mejor, cuanto mas prevista es la operación incorrecta de usuario
Tipo de escala	Absoluta
Tipo de medida	A = Cantidad B = Cantidad X = Cantidad/ Cantidad
Entrada para la medición	Reporte de pruebas Reporte de operación
Referencia PCVS ISO/IEC 12207	5.3 Integración 5.3 Pruebas de calificación 5.4 Operación
Audiencia objetivo	Usuario Responsable de mantenimiento

Tabla C.21: Métrica de tolerancia a fallos - prevención de operación incorrecta

C.9 METRICAS DE USABILIDAD

C.9.1 METRICA DE ENTENDIBILIDAD

Nombre	**Comprensión de entradas y salidas**
Propósito de la métrica	¿Pueden los usuarios entender que es requerido como datos de entrada y que es proporcionado como salida por el sistema software?
Método de aplicación	Conducir la prueba de usuario y entrevistar al usuario con los cuestionarios u observar su comportamiento. Contar el número de los datos entrada y salida de datos entendidos por el usuario y comparar con el número total de estos disponibles para el usuario.
Medición, fórmula	X = 1 - A/B A = Número de datos de entrada y salida entendidos satisfactoriamente por el usuario. B = Número de datos de entrada y salida disponible de la interfaz

Interpretación del valor medido	0 <= X <= 1, Lo más cercano a 1,0 es lo mejor
Tipo de escala	Absolua
Tipo de medida	A = Cantidad B = Cantidad X = Cantidad/ Cantidad
Entrada para la medición	Manual de usuario Reporte de operación
Referencia PCVS ISO/IEC 12207	6.5 Validación 5.3 Prueba de la calificación 5.4 Operación
Audiencia objetivo	Usuario Responsable de mantenimiento

Tabla C.22: Métrica de entendibilidad – comprensión de entradas y salidas

C.9.2 METRICA DE FACILIDAD DE APRENDIZAJE

Nombre	**facilidad de aprender a realizar una tarea en uso**
Propósito de la métrica	¿Cuánto tiempo el usuario toma para aprender cómo realizar la tarea especificada eficazmente?
Método de aplicación	Observar el comportamiento de los usuarios desde que comienzan a aprender hasta que ellos comienzan a operar eficazmente
Medición, fórmula	T = Suma de tiempo de la operación del usuario hasta que el usuario logra realizar la tarea especificada en un tiempo corto
Interpretación del valor medido	0 < T, El valor mas pequeño es lo mejor
Tipo de escala	Ratio
Tipo de medida	T = Tiempo
Entrada para la medición	Reporte de operación Registro del monitoreo al usuario
Referencia PCVS ISO/IEC 12207	6.5 Validación 5.3 Prueba de la calificación 5.4 Operación
Audiencia objetivo	Usuario Responsable de mantenimiento

Tabla C.23: Métrica de facilidad de aprendizaje - facilidad de aprender para realizar una tarea en uso

C.9.3 METRICA DE OPERABILIDAD

Nombre	**entendibilidad del mensaje en uso**
Propósito de la métrica	¿Puede el usuario entender fácilmente los mensajes de sistema software? ¿Hay algún mensaje cuyo entendimiento causa un retraso antes de comenzar la siguiente acción? ¿Puede el usuario memorizar fácilmente mensajes importantes?
Método de aplicación	Observar el comportamiento de usuario que este operando el sistema
Medición, fórmula	X = A/TUO A = Número de veces que el usuario se detiene por un período largo o sucesivamente, y repetidamente falla en la misma operación, por

	dificultad para comprender un mensaje TUO= tiempo de operación del usuario (período de observación)
Interpretación del valor medido	0 <= X, El valor más pequeño y cercano a 0,o es lo mejor
Tipo de escala	Ratio
Tipo de medida	A = Cantidad TUO = Tiempo X = Cantidad/ Tiempo
Entrada para la medición	Reporte de operación Registro del monitoreo al usuario
Referencia PCVS ISO/IEC 12207	6.5 Validación 5.3 Prueba de la calificación 5.4 Operación
Audiencia objetivo	Usuario Diseñador de interfaz de usuario

Tabla C.24: Métrica de operabilidad - entendibilidad del mensaje en uso

C.10 METRICAS DE EFICIENCIA

C.10.1 METRICA DE COMPORTAMIENTO EN EL TIEMPO

Nombre	**rendimiento**
Propósito de la métrica	¿Cuántas tareas pueden ser ejecutarse satisfactoriamente en un determinado intervalo de tiempo?
Método de aplicación	Calibrar cada tarea de acuerdo a la prioridad pre establecida. Iniciar varias tareas de trabajo. Medir el tiempo que toma completar la operación de las tareas medidas. Mantener un registro de cada intento.
Medición, fórmula	X = A/T A = Número de tarea completadas T = Intervalo de tiempo de observación
Interpretación del valor medido	0 < X, El mayor valor es lo mejor
Tipo de escala	Ratio
Tipo de medida	A = Cantidad T = Tiempo X = Cantidad/Tiempo
Entrada para la medición	Reporte de prueba Registro de operación mostrando tiempo trascurrido
Referencia PCVS ISO/IEC 12207	5.3 Integración de sistema 5.3 Integración del software 5.3 Pruebas de calificación 5.4 Operación 5.5 Mantenimiento
Audiencia objetivo	Usuario Desarrollador Responsable de mantenimiento Responsable de ACS

Tabla C.25: Métrica de comportamiento en el tiempo - rendimiento

C.11 METRICAS DE FACILIDAD DE MANTENIMIENTO

C.11.1 METRICA DE ANALIZABILIDAD

Nombre	**soporte a las funciones de diagnóstico**
Propósito de la métrica	¿Cuán capaces son las funciones de diagnóstico en el soporte del análisis de causa? ¿Puede el usuario identificar la operación específica que causo falla? (El usuario es capaz de evitar caer nuevamente en la ocurrencia de fallar con una operación alternativa) ¿Puede el responsable de mantenimiento fácilmente encontrar la causa de la falla?
Método de aplicación	Observar el comportamiento del usuario o responsable de mantenimiento quienes están tratando de resolver las fallas utilizando funciones de diagnóstico
Medición, fórmula	X = A/B A = Número de fallos que el responsable de mantenimiento puede diagnosticar (usando funciones de diagnóstico) para entender la relación causa-efecto B = Número total de fallas registrados
Interpretación del valor medido	0 <= X <= 1, El más cercano a 1,0 es lo mejor
Tipo de escala	Absoluta
Tipo de medida	A = Cantidad B = Cantidad X = Cantidad/Cantidad
Entrada para la medición	Reporte de resolución de problema Reporte de operación
Referencia PCVS ISO/IEC 12207	5.3 Pruebas de calificación 5.4 Operación 5.5 Mantenimiento
Audiencia objetivo	Desarrollador Responsable de mantenimiento Operador

Tabla C.26: Métrica de analizabilidad - soporte a las funciones de diagnóstico

C.12 METRICAS DE PORTABILIDAD

C.12.1 METRICA DE INSTALABILIDAD

Nombre	**facilidad de instalación**
Propósito de la métrica	¿Puede el usuario o responsable de mantenimiento instalar fácilmente el software en el entorno de operación?
Método de aplicación	Observar el comportamiento del usuario o del responsable de mantenimiento cuando el usuario está tratando de instalar el software en el ambiente de operación
Medición, fórmula	X = A/B A = El número de casos en los cuales un usuario exitosamente cambio la instalación a su conveniencia B = Número total de casos en el que un usuario intento cambiar la instalación a su conveniencia
Interpretación del valor medido	0 <= X <= 1, El más cercano a 1,0 es lo mejor
Tipo de escala	Absoluta
Tipo de	A = Cantidad

medida	B = Cantidad X = Cantidad/Cantidad
Entrada para la medición	Reporte de resolución de problema Reporte de operación
Referencia PCVS ISO/IEC 12207	5.3 Pruebas de calificación 5.4 Operación 5.5 Mantenimiento
Audiencia objetivo	Desarrollador Responsable de mantenimiento Operador

Tabla C.27: Métrica de instalabilidad - facilidad de instalación

ANEXO D

MEDICION DE LA CALIDAD INTERNA Y EXTERNA DEL PRODUCTO SOFTWARE

D.1 MEDIDAS PARA LA CALIDAD DEL PRODUCTO SOFTWARE - ICONIX

APLICABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Adecuación Funcional	¿Cuan adecuada son las funciones revisadas?	X = 1-A/B A = número de métodos inadecuados. B = numero de métodos revisados.	75.00%	0 <= X <= 1 Entre más cercano a 1, más adecuado.	48	192	0.75
Integridad de implementación funcional	¿Cuan completa es la implementación de acuerdo a la especificación de requerimientos?	X = 1-A/B A= número de métodos no implementados B= número total métodos incluyendo los no implementados	81.82%	0 <= X <= 1, lo mas cerca de 1,0 es lo mejor	48	264	0.818182
cobertura de la implementación funcional	¿Cuán correcta es la implementación funcional?	X = 1-A/B A= número de métodos que faltan mas métodos inadecuados B= total de métodos de los requisitos	63.64%	0 <= X <= 1 Entre más cercano a 1, más adecuado.	96	264	0.636364
			73.48%	**86.62%**			

PRECISION

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Exactitud de cálculo	¿Cuán completamente se implementaron los requerimientos de exactitud?	X = A/B A= número de métodos implementados con exactitud de calculo B= número total de métodos que esperan exactitud de calculo.	72.92%	0 <= X <= 1, Lo más cercano a 1 es lo mejor	35	48	0.729167
			72.92%	**70.00%**			

Tabla D.1: Medida de la calidad del producto para la funcionalidad

MADUREZ

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	Y
Eliminación de fallas	¿Cuántas fallas fueron corregidas? ¿Cuál es la proporción de fallas removidas?	Y = A/B A = número de fallas corregidas en codificación. B = número de fallas detectadas en las revisión.	75.00%	0<=Y<=1, Mientras mas cercano a 1 mejor (mas fallas removidas)	72	96	0.75
			75.00%	**84.44%**			

TOLERANCIA A FALLAS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Prevención de operación incorrecta	¿Cuántos funciones se han implementado con capacidad de prevención de operación incorrectas?	X = A/B A = número de funciones implementadas para evitar patrones de operación incorrecta. B = número de patrones de operación incorrecta que deben considerarse. (datos incorrectos, secuencia de datos incorrectos y secuencia de operacion incorrecta) Comentarios.- Patrones de operación incorrecta son; tipos de datos incorrectos como parámetros, secuencia de datos de entrada incorrecta, secuencia de operación incorrecta.	53.70%	0 <= X , Cuando "X" es mayor mejor es la prevención de operación incorrecta	29	54	0.53703704
			53.70%	**71.43%**			

Tabla D.2: Medida de la calidad del producto para la fiabilidad

ENTENDIBILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
función de comprensión	¿Qué proporción de las funciones del producto será el usuario capaz de entender en forma correcta?	X = A/B A = número de funciones presentes en las interfaces de los usuarios cuyo propósito es entendido por el usuario. B = n° total de funciones presentes en las interfaces de los usuarios.	66.67%	0 <= X <=1, Lo más cerca de 1 es lo mejor	48	72	0.66667
			66.67%	**85.00%**			

FACILIDAD DE APRENDIZAJE

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Integridad de la documentación del usuario y/o facilidad de ayuda	¿Qué proporción de las funciones son descritas en la documentación para el usuario y/o facilidades de ayuda?	X = A/B A = número de funciones implementadas con ayuda. B = número total de funciones que requieren ayuda	50.00%	0 <= X <=1, Lo cercano a 1 es lo mejor	72	144	0.5
			50.00%	**64.00%**			

OPERABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Claridad de mensajes	¿Qué proporción de los mensajes son auto-explicativos?	X = A/B A = n° de mensajes implementados con explicaciones claras B = número total de mensajes implementados	58.33%	0 <= X <=1, Lo mas cercano a 1 es lo mejor	28	48	0.58333
Claridad de la interfaz	¿Qué proporción de los elementos de la interfaz son auto-explicativos?	X = A/B A = número de elementos de interfaz que son auto explicativos B = n° total de elementos de interfaz	80.00%	0 <= X <=1, Lo mas cercano a 1 es lo mejor	96	120	0.8
			69.17%	**66.67%**			

Tabla D.3: Medida de la calidad del producto para la usabilidad

UTILIZACIÓN DE RECURSOS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos	
					Tamaño Kb	X
Utilización de memoria	¿Cuál es el tamaño estimado de memoria que ocupará el producto para completar una tarea específica?	X = Tamaño en bytes (calculado o simulado) (512 Kb - 2048 Kb)	81.25%	Lo menor es lo mejor	800	0.8125
			81.25%			

COMPORTAMIENTO DE TIEMPOS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Rendimiento	¿Cuántas tareas pueden ser ejecutarse satisfactoriamente en un determinado intervalo de tiempo?	X = A/T A = Número de tarea completadas (cu) T = Intervalo de tiempo de observación (min)	68.18%	0 < X, El mayor valor es lo mejor	1	5.5	0.681818
			68.18%				

Tabla D.4: Medida de la calidad del producto para la eficiencia

CAMBIABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Registro de cambios	¿Son los cambios a las especificaciones y módulos de programa son registrados adecuadamente en el código y haciendo uso de comentarios?	X = A/B A = número de cambios en métodos que tienen comentarios, confirmados en la revisión. B = número total de metodos alterados desde la primera versión del código.	60.00%	0 <= X <= 1 Lo mas cerca a 1, indica un mayor registro. Cuando el control de cambio indica 0, significa un pobre control de cambios o pequeños cambios, alta estabilidad.	72	120	0.6
			60.00%				

Tabla D.5: Medida de la calidad del producto para la facilidad de mantenimiento

ADAPTABILIDAD

Nombre	Propósito	Método de Aplicación	Medición / Fórmula	Resultado	Interpretación
adaptabilidad al entorno organizacional (adaptabilidad a la organización y a la estructura de la misma)	¿Cuán adaptable es el producto al cambio organizacional ?	Contar el número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en organizaciones múltiples según lo especificado y comparar con el número de funciones con requisitos de adaptabilidad al entorno organizacional.	X = A/B A = Número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en el ambiente de organizaciones y de negocio múltiples según lo especificado, confirmado en la revisión. B = Número total de funciones con requisitos de adaptabilidad al ambiente de la organización. Obs.-El software para comercializar tara, es una aplicacion web, que tiene una arquitectura técnica definida, por tanto no existe problemas de adaptabilidad.	100.00%	0 <= X <= 1, Lo mas cercano a 1 es lo mejor
				100.00%	

INSTALABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
facilidad de reinstalación	¿Puede el usuario o responsable de mantenimiento fácilmente reinstalar el software?	X = 1-A/B A = El número de casos en los cuales un usuario falla al reinstalar el software B = Número total de casos en que un usuario intenta reinstalar el software	80.00%	0 <= X <= 1, El más cercano a 1,0 es lo mejor	1	5	0.8
			80.00%				

Tabla D.6: Medida de la calidad del producto para la portabilidad

D.2 MEDIDAS PARA LA CALIDAD DE PRODUCTO SOFTWARE - XP

APLICABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Adecuación Funcional	¿Cuan adecuada son las funciones revisadas?	X = 1-A/B A = número de metodos inadecuados. B = numero de metodos revisados.	80.00%	0 <= X <= 1 Entre más cercano a 1, más adecuado.	48	240	0.8
Integridad de implementación funcional	¿Cuan completa es la implementación de acuerdo a la especificación de requerimientos?	X = 1-A/B A= número de metodos no implementados B= número total metodos incluyendo los no implementados	84.62%	0 <= X <= 1, lo mas cerca de 1,0 es lo mejor	48	312	0.846154
cobertura de la implementación funcional	¿Cuán correcta es la implementación funcional?	X = 1-A/B A= número de metodos que faltan mas metodos inadecuados B= número total de metodos de los requisitos	69.23%	0 <= X <= 1 Entre más cercano a 1, más adecuado.	96	312	0.692308
			77.95%	**85.09%**			

PRECISION

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Exactitud de cálculo	¿Cuán completamente se implementaron los requerimientos de exactitud?	X = A/B A= número de metodos implementados con exactitud de calculo B= número total de metodos que esperan exactitud de calculo.	62.50%	0 <= X <= 1, Lo más cercano a 1 es lo mejor	30	48	0.625
			62.50%	**78.33%**			

Tabla D.7: Medida de la calidad del producto para la funcionalidad

MADUREZ

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	Y
Eliminación de fallas	¿Cuántas fallas fueron corregidas? ¿Cuál es la proporción de fallas removidas?	Y = A/B A = número de fallas corregidas en codificación. B = número de fallas detectadas en las revisión.	66.67%	0<=Y<=1, Mientras mas cercano a 1 mejor (mas fallas removidas)	48	72	0.66666667
			66.67%	**79.17%**			

TOLERANCIA A FALLOS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Prevención de operación incorrecta	¿Cuántos funciones se han implementado con capacidad de prevención de operación incorrectas?	X = A/B A = número de funciones implementadas para evitar patrones de operación incorrecta. B = número de patrones de operación incorrecta que deben considerarse. (datos incorrectos, secuencia de datos incorrectos y secuencia de operacion incorrecta) Comentarios.- Patrones de operación incorrecta son; tipos de datos incorrectos como parámetros, secuencia de datos de entrada incorrecta, secuencia de operación incorrecta.	34.88%	0 <= X , Cuando "X" es mayor mejor es la prevención de operación incorrecta	15	43	0.34883721
			34.88%	**66.17%**			

Tabla D.8: Medida de la calidad del producto para la fiabilidad

ENTENDIBILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
función de comprensión	¿Qué proporción de las funciones del producto será el usuario capaz de entender en forma correcta?	X = A/B A = número de funciones presentes en las interfaces de los usuarios cuyo propósito es entendido por el usuario. B = número total de funciones presentes en las interfaces de los usuarios.	66.67%	0 <= X <=1, Lo más cerca de 1 es lo mejor	72	108	0.66667
			66.67%	**80.95%**			

FACILIDAD DE APRENDIZAJE

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Integridad de la documentación del usuario y/o facilidad de ayuda	¿Qué proporción de las funciones son descritas en la documentación para el usuario y/o facilidades de ayuda?	X = A/B A = número de funciones implementadas con ayuda. B = número total de funciones que requieren ayuda	75.00%	0 <= X <=1, Lo cercano a 1 es lo mejor	144	192	0.75
			75.00%	**67.50%**			

OPERABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Claridad de mensajes	¿Qué proporción de los mensajes son auto-explicativos?	X = A/B A = número de mensajes implementados con explicaciones claras B = número total de mensajes implementados	58.33%	0 <= X <=1, Lo mas cercano a 1 es lo mejor	28	48	0.58333
Claridad de la interfaz	¿Qué proporción de los elementos de la interfaz son auto-explicativos?	X = A/B A = número de elementos de interfaz que son auto explicativos B = número total de elementos de interfaz	83.33%	0 <= X <=1, Lo mas cercano a 1 es lo mejor	150	180	0.83333
			70.83%	**63.89%**			

Tabla D.9: Medida de la calidad del producto para la usabilidad

UTILIZACIÓN DE RECURSOS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos	
					Tamaño Kb	X
Utilización de memoria	¿Cuál es el tamaño estimado de memoria que ocupará el producto para completar una tarea especifica?	X = Tamaño en bytes (calculado o simulado) (512 Kb - 2048 Kb)	66.67%	Lo menor es lo mejor	1024	0.6667
			66.67%			

COMPORTAMIENTO DE TIEMPOS

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Rendimiento	¿Cuántas tareas pueden ser ejecutarse satisfactoriamente en un determinado intervalo de tiempo?	X = A/T A = Número de tarea completadas (hu) T = Intervalo de tiempo de observación (min)	57.02%	0 < X, El mayor valor es lo mejor	1	5.7	0.57
			57.02%				

Tabla D.10: Medida de la calidad del producto para la eficiencia

CAMBIABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
Registro de cambios	¿Son los cambios a las especificaciones y módulos de programa son registrados adecuadamente en el código y haciendo uso de comentarios?	X = A/B A = número de cambios en metodos que tienen comentarios, confirmados en la revisión. B = número total de metodos alterados desde la primera versión del código.	66.67%	0 <= X <= 1 Lo mas cerca a 1, indica un mayor registro. Cuando el control de cambio indica 0, significa un pobre control de cambios o pequeños cambios, alta estabilidad.	96	144	0.667
			66.67%				

Tabla D.11: Medida de la calidad del producto para la capacidad de mantenimiento

ADAPTABILIDAD

Nombre	Propósito	Método de Aplicación	Medición / Fórmula	Resultado	Interpretación
adaptabilidad al entorno organizacional (adaptabilidad a la organización y a la estructura de la misma)	¿Cuán adaptable es el producto al cambio organizacional?	Contar el número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en organizaciones múltiples según lo especificado y comparar con el número de funciones con requisitos de adaptabilidad al entorno organizacional.	X = A/B A = Número de las funciones implementadas que son capaces de alcanzar los resultados requeridos en el ambiente de organizaciones y de negocio múltiples según lo especificado, confirmado en la revisión. B = Número total de funciones con requisitos de adaptabilidad al ambiente de la organización. Obs.-El software para comercializar tara, es una aplicacion web, que tiene una arquitectura técnica definida, por tanto no existe problemas de adaptabilidad.	100.00%	0 <= X <= 1, Lo mas cercano a 1 es lo mejor
				100.00%	

INSTALABILIDAD

Nombre	Propósito	Medición / Fórmula	Resultado	Interpretación	Valores Obtenidos		
					A	B	X
facilidad de reinstalación	¿Puede el usuario o responsable de mantenimiento fácilmente reinstalar el software?	X = 1-A/B A = El número de casos en los cuales un usuario falla al reinstalar el software B = Número total de casos en que un usuario intenta reinstalar el software	75.00%	0 <= X <= 1, El más cercano a 1,0 es lo mejor	1	4	0.75
			75.00%				

Tabla D.12: Medida de la calidad del producto para la portabilidad

ANEXO E

INSTRUMENTO PARA LA RECOLECCION DE INFORMACION SOBRE COMERCIALIZACION DE TARA EN LA REGION AYACUCHO

Aspecto	Huanta	Huamanga	Cangallo	La Mar
Rangos de altitud (msnm)				
Centros poblados (provincia, distrito, y comunidad)				
Epocas de lluvia y sequía				
Temperaturas promedio (°C)				
Riesgos climáticos principales				
Fuentes de agua				
Tipo de suelo				
Tierra con riego				
Superficie de producción (Ha.)				
Epocas de cosecha (mes)				
Volumen promedio cosecha (Tn. por mes)				

Tabla F.1: Matriz de caracterización de producción.

Eslabón	Actores	Tipo		Ubicación
		Directo	Indirecto	
Producción				
Acopio				
Transformación				
Comercialización				

Tabla F.2: Inventario de actores directos e indirectos

Características	Eslabón			
	Producción	Acopio	Transfor-mación	Comercia-lización
Actores. ¿Quiénes son?				
Actividades.¿Qué hace?				
Estrategia. ¿Cómo lo hace?				
Cultura. ¿Qué costumbres tiene?				
Organización. ¿Cómo es su organización?				

Resultados económicos. ¿Qué resultados tiene?				

Tabla F.3: Caracterización de actores directos

Características	Eslabón			
	Producción	Acopio	Transfor-mación	Comercia-lización
Actores. ¿Quiénes son?				
Tipo. ¿Qué naturaleza tiene?				
Actividades. ¿Qué bienes, servicios o insumos oferta?				
Clientes. ¿Quiénes son sus clientes?				
Estrategia. ¿Qué modalidad aplica para ofertar sus bienes, servicios o insumos?				
Requisitos. ¿Qué requisitos aplica para acceder a sus bienes, servicios o insumos?				
Cobertura. ¿Qué capacidad de atención tiene?				
Calidad. ¿Qué calidad tienen sus bienes, servicios o insumos?				
Resultados económicos. ¿Qué costos tienen sus bienes, servicios o insumos?				

Tabla F.4: Caracterización de actores indirectos

Relaciones de Acopladores Locales	Productor		Acopiador ...	
	Con Productores Organizados	Con Productores Individuales	Con Acopiadores Regionales	Con Acopiadores de Lima
Frecuencia de la Transacción				
Formalidad de la Transacción				
Poder de negociación. ¿Quién define el precio?				
Modalidad de pago				

Tabla F.5: Relaciones entre actores directos

Característica / Zona	Huanta	Huamanga	Cangallo	La Mar
Superficie (Ha.)				
Sistema de producción				
Productividad				
Calidad				
Precios (S/.Kg.)				
Costos				
Época de cosecha				
Mercados				

Tabla F.6: Matriz de competencia

Mercado	Tipo de mercado	Meses de mercado	Modalidad de pago	Precio (S./Kg)	Provee-dor	Volumen (Tn)
Mercado 1						
Mercado 2						
Mercado 3						
Mercado n						

Tabla F.7: Matriz de mercados

Años	Precio(S./Kg) por Provincia			
	Huanta	Huamanga	Cangallo	La Mar
1990				
1995				
2000				
2002				
2004				
2005				
2006...				

Tabla F.8: Matriz histórica de precios

ANEXO F
GLOSARIO DE TERMINOS DE CALIDAD DE SOFTWARE

Adquiriente

Una organización que adquiere u obtiene un sistema, producto de software o servicio software de un proveedor.

Atributo

Una característica física o abstracta mensurable de una entidad. Los atributos pueden ser internos o externos.

Calidad

Son todas las características de una entidad que forman parte de su habilidad para satisfacer las necesidades propias e implícitas.

Calidad en uso

Medida en que un producto usado por usuarios específicos satisface sus necesidades para lograr metas especificas con eficacia, productividad y satisfacción.

Calidad externa

La extensión para la cual un producto satisface necesidades explícitas e implícitas cuando es usado bajo condiciones específicas.

Calidad interna

Es la totalidad de atributos del producto que determinan su habilidad para satisfacer las necesidades propias e implícitas bajo condiciones específicas.

Calificación

La acción de evaluar el valor medido al nivel de calificación adecuado. Utilizado para determinar el nivel de calificación asociado con el software para una característica específica de calidad.

Defecto

Un paso, proceso o definición de dato incorrecto en un programa de computadora.

Desarrollador

Una organización que realiza actividades de desarrollo, durante el proceso del ciclo de vida del software.

Escala

Un conjunto de valores con propiedades definidas

Ejemplos de tipos de escalas son: una escala nominal que corresponda a un conjunto de categorías; una escala ordinal que corresponda a un conjunto ordenado de puntos; una escala de intervalo que corresponda a una escala ordenada con puntos equidistantes; y una escala de ratios que no sólo tiene puntos equidistantes sino que posee el cero absoluto. Las métricas utilizando escalas nominales u ordinales producen datos cualitativos, y las métricas utilizando escalas de intervalos o ratios producen datos cuantitativos.

Evaluación de Calidad

Es un examen sistemático del grado o capacidad de una entidad para satisfacer necesidades o requerimientos específicos.

Falla

La terminación de la capacidad de un producto de realizar una función requerida o su incapacidad para realizarla dentro de límites previamente especificados.

Indicador

Una medida que se puede utilizar para estimar o para predecir otra medida. Los indicadores pueden emplearse para evaluar los atributos cualitativos del software y para calcular los atributos del proceso de desarrollo. Ambos son valores indirectos e imprecisos de los atributos.

Medición

Actividad que usa la definición de la métrica para producir el valor de una medida.

Es el proceso por el cual los números o símbolos son asignados a atributos o entidades en el mundo real tal como son descritos de acuerdo a reglas claramente definidas. (Fenton, 1991)

Medida (sustantivo)

Número o categoría asignada a un atributo de una entidad al realizar una medición.

Proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. (Pressman, 2002)

Medida directa

Una medida de un atributo que no depende de la medida de ningún otro atributo.

Medida externa

Una medida indirecta de un producto derivada de las medidas del comportamiento del sistema del que es parte.

El sistema incluye cualquier hardware, software (ya sea software a medida o software tipo paquete) y usuarios.

El número de fallas encontradas durante las pruebas es una medida externa del número de fallas en el programa, porque el número de fallas es contado durante la operación del programa corriendo en un sistema de cómputo.

Las medidas externas pueden ser usadas para evaluar los atributos de calidad cercanos a los objetivos finales de diseño.

Medir (verbo)

Número o valor que una entidad le asigna a un atributo al efectuar una medición.

Métrica

Es un método definido de valoración y su escala de valoración.

Las métricas pueden ser internas o externas, directas o indirectas.

Las métricas incluyen métodos para clasificar la data o información cualitativa en diferentes categorías.

Modelo cualitativo

Es una serie de características y la relación entre las mismas, que conforman la base de los requerimientos cualitativos específicos y la valoración cualitativa.

Módulo de evaluación

Un paquete de tecnología de evaluación para una característica o sub característica de calidad de un software específico. El paquete incluye métodos y técnicas de evaluación, entradas a ser evaluadas, datos a ser medidos y recopilados y procedimientos y herramientas de soporte.

Necesidades implícitas

Necesidades que pueden no haber sido especificadas pero que son necesidades reales cuando la entidad es usada en condiciones particulares. Necesidades implícitas son necesidades reales, las cuales pueden no haber sido documentadas.

Nivel de calificación

Un punto en la escala ordinal que es utilizado para categorizar una escala de medida.

El nivel de calificación habilita al software para ser clasificado de acuerdo con las necesidades explícitas o implícitas.

Los niveles de clasificación adecuados pueden ser asociados con las vistas diferentes de calidad, por ejemplo, usuarios, gerentes o desarrolladores.

Producto software

El conjunto de programas de cómputo, procedimientos, y posible documentación y datos asociados.

Los productos incluyen productos intermedios y productos para los usuarios, como los desarrolladores y personal de soporte.

Producto de software intermedio

Es un producto del proceso de desarrollo del software que se emplea para alimentar una etapa diferente del proceso de desarrollo.

En algunos casos, un producto intermedio puede ser también un producto final.

Proveedor

Una organización que entra a un contrato con el adquiriente para el suministro de un sistema, producto de software o servicio de software bajo los términos de dicho contrato.

Servicio

Es una organización que presta servicios de mantenimiento.

Sistema

Una composición integrada que consiste en uno o más procesos, hardware, software, instalaciones y personas, que proveen una capacidad para satisfacer una necesidad establecida o un objetivo.

Software

Todo o parte de los programas, procedimientos, reglas y documentación asociada a un sistema de procesamiento de información.

El software es una creación intelectual que es independiente del medio en el cual fue grabado.

Usuario

Un individuo que utiliza el producto de software para realizar una función específica.

Los usuarios pueden incluir operadores, receptores de los resultados del software, desarrolladores o personal de soporte de software.

Validación

Confirmación por inspección y provisión de evidencia objetiva de que los requerimientos particulares para un uso específico son alcanzados.

En diseño y desarrollo, la validación está relacionada con el proceso de reexaminación de un producto para determinar la conformidad con las necesidades del usuario.

La validación es realizada normalmente sobre el producto final bajo condiciones operacionales definidas. Puede ser necesaria en las fases iniciales.

"Validado" es utilizado para designar el estado correspondiente.

Valoración

Emplear una métrica para asignar uno de los valores de una escala (el mismo que puede ser un número o categoría) al atributo de una entidad.

La valoración puede ser cualitativa cuando se emplean categorías. Por ejemplo, algunos de los atributos importantes de los productos de software, tales como el lenguaje del programa base (ADA, C, COBOL, etc.) son categorías cualitativas.

Valoración indirecta

Es la valoración de un atributo derivada del valor de uno o más atributos diferentes. La valoración externa de un atributo de un sistema de cómputo (tal como el tiempo de respuesta a la información alimentada por el usuario) es una valoración indirecta de los atributos del software, dado que esta medida se verá influenciada por los atributos del entorno de cómputo, así como por los atributos propios del software.

Valoración interna

Es una valoración del producto en sí, ya sea directa o indirecta.

El número de líneas del código, las valoraciones de complejidad, el número de fallas encontradas durante el proceso y el índice de señales o alertas, son todas las valoraciones internas propias del producto en sí.

Valorar (verbo)

Realizar una valoración o estimación.

Valor (sustantivo)

Es el número o categoría que una entidad le asigna a un atributo al efectuar la valoración.

Valoración Cualitativa

Es una evaluación sistemática del grado o capacidad de una entidad para satisfacer necesidades o requerimientos específicos.

Dichos requerimientos pueden ser formalmente especificados, por ejemplo, por el área de desarrollo de sistemas, cuando el producto se diseña por contrato para un usuario específico, cuando el producto es desarrollado sin un usuario específico, o bien que se trate de necesidades más generales, como cuando un usuario evalúa los productos con propósitos de comparación y selección.

Verificación

Confirmación por examen y provisión de evidencia objetiva que los requerimientos específicos han sido alcanzados.

En diseño y desarrollo, la verificación está relacionada con el proceso de examinar el resultado de una actividad dada para determinar su conformidad con los requerimientos definidos para dicha actividad.

"Verificado" es utilizado para designar el estado correspondiente.