UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

DIVISIÓN DE ESTUDIOS DE POSGRADO

MAESTRÍA EN ELECTRÓNICA Y COMPUTACIÓN

"DESARROLLO DE UNA METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMPOTRADOS BAJO EL PARADIGMA DE MEJORA DEL PROCESO SOFTWARE"

TESIS

Para obtener el grado de Maestro en Electrónica y Computación

PRESENTA
Ing. Andrea Ismeneé Herrera Huerta

DIRECTOR DE TESIS
Dr. Iván Antonio García Pacheco

Heroica Ciudad de Huajuapan de León, Oaxaca. Agosto 2011

Tesis presentada el 12 de Agosto de 2011 ante los siguientes sinodales:

Dr. ENRIQUE GUZMÁN RAMÍREZ
Dr. FELIPE DE JESÚS TRUJILLO ROMERO
M.C. FELIPE SANTIAGO ESPINOSA
M.C. MARIBEL TELLO BELLO

Bajo la dirección de:

Dr. IVÁN ANTONIO GARCÍA PACHECO

ÍNDICE

ÍNDICE

ÍNDICE

ÍNDICE

ÍNDICE

Figuras

ÍNDICE

Tablas

1

INTRODUCCIÓN Y DEFINICIÓN DEL PROBLEMA

Desde el 2002, los Sistemas Empotrados (SE) están cada vez más presentes en la sociedad [Solingen, 2002], abarcando diversas áreas, como el transporte, la electrónica de consumo, la distribución de energía, la medicina, el control industrial, las estaciones de red, entre otras [Noergaard, 2005]. De acuerdo a [Graaf, Lormans & Toetenel, 2002] y [Noergaard, 2005], el campo de los SE es cada vez más amplio, variado y complejo; por lo que las inversiones en la industria de las tecnologías de Ingeniería de Software Empotrado para el desarrollo de este tipo de sistemas tenderán a aumentar considerablemente.

Regularmente es difícil describir a los SE ya que se encuentran en constante evolución por los avances tecnológicos, sin embargo, las descripciones coinciden en que se trata de "*una mezcla de hardware y software dedicado a una aplicación especifica*" [Graaf, Lormans & Toetenel, 2002]; que además "*implica un desarrollo paralelo de software y de hardware*" [Berger, 2002]. Su diseño se basa en los modelos clásicos de la Ingeniería de Sistemas (Bing-bang model, cascada, espiral, Code-and-fix model, etc.) y en dos elementos conceptuales: la arquitectura de hardware y la arquitectura de software. Lo anterior origina que el desarrollo de los SE sea interdisciplinario [Pearson, Giuma & Harris, 2008], involucrando así a ingenieros y técnicos especializados tanto en el diseño de hardware como en el diseño de software.

Las necesidades de las sociedades actuales, han ocasionado que los SE incrementen su complejidad y que por ende se destine mayor importancia al tiempo de desarrollo de los mismos. Ahora bien, durante el proceso de desarrollo es necesario considerar que los requisitos de los SE son únicos e impactan directamente sobre los métodos y herramientas utilizadas para su desarrollo [Ibrahim, Zhao & Kinghorn, 2006], además de la importancia que el software tiene en la funcionalidad del sistema [Liggesmeyer & Trapp, 2009].

El cambio constante en los requisitos y la complejidad variante provocaron que las metodologías de diseño actuales ya no fueran suficientes [Sangiovanni & Martin, 2001], pues los sistemas modernos exigían nuevos enfoques para su especificación, diseño y desarrollo. Debido a esto, las organizaciones que participan en el diseño, desarrollo y logística de los SE deben cambiar fundamentalmente la manera en que los desarrollan, gestionando adecuadamente la complejidad operativa y ambiental [Srinivasan, Dobrin & Lundqvist, 2009].

Con el objetivo de aumentar la calidad del producto, reducir el tiempo de comercialización, y reducir los costos de desarrollo, en [Liggesmeyer & Trapp, 2009] se recomienda que las metodologías empleadas para el desarrollo de SE deben integrar correctamente los paradigmas esenciales del *diseño de software y hardware*, evitando así los problemas que actualmente se presentan durante la etapa de integración [Srinivasan, Dobrin & Lundqvist, 2009].

De acuerdo a [Srinivasan, Dobrin & Lundqvist, 2009], las metodologías utilizadas actualmente en la industria de los SE no son específicas del dominio de éstos y no se encuentran estandarizadas. Es decir, estos sistemas carecen de una metodología eficaz por lo que en la industria, la sensación general es que la práctica actual de desarrollo de software y SE es insatisfactoria.

En este sentido, Graaf, Lormans y Toetenel estudiaron a siete empresas europeas en [Graaf, Lormans & Toetenel, 2003] para determinar el estado de la práctica de la Ingeniería de Software Empotrado. Una de las conclusiones clave del estudio fue que "*los sistemas de decisión de ingeniería son en gran parte impulsados por las limitaciones de hardware, que a su vez, impactan los esfuerzos de software en dos etapas del ciclo de vida cuando se desarrollan los requisitos de software a nivel componente*". A pesar de que el estudio no indica de forma concisa cuáles son estas dos etapas del ciclo de vida que son afectadas, los datos ofrecidos en el estudio permiten identificar que se refieren a la *Especificación de Requisitos y Diseño*. Sin embargo, es claro que desde el 2000 han existido propuestas que intentan agregar formalidad al proceso de desarrollo. Algunas de estas propuestas han proporcionado metodologías para el desarrollo de SE, tales como: PROFES [Birk et al., 1998], BOOTSTRAP [Kuvaja & Bicego, 1994], REMES [Seceleanu, Vulgarakis & Petterson, 2009], PECOS [Genßler et. al., 2002], POLIS [Kang, Kwon & Lee, 2005], Métodos Ágiles [Greene, 2004], entre otros.

Desafortunadamente, las metodologías disponibles para el desarrollo de software (adaptadas al desarrollo de SE) no consideran las necesidades específicas de este tipo de sistemas [Srinivasan, Dobrin & Lundqvist, 2009].

Por todo lo anteriormente expuesto, resulta evidente que el problema actual de los SE no es la falta de normas o modelos para su desarrollo, sino la falta de una estrategia eficaz para aplicarlas con éxito. Si a esto le sumamos la extensa variedad de herramientas y métodos para desarrollar los SE, el problema se dificulta todavía más. Por lo que resulta necesario establecer una metodología de desarrollo propia para los SE; que considere la estrecha relación que existe entre el software y el hardware; que proporcione una fácil integración en el menor tiempo posible, y que refleje la calidad adecuada.

1.1. IMPORTANCIA DEL PROBLEMA

En la actualidad, la relevancia que están adquiriendo los SE es indiscutible. Un estudio realizado por Aberdeen Group en el año 2000 [URL-1], reveló que el 66% de productos nuevos desarrollados incluían un SE. Así mismo, en el 2003 se pronosticó que en los próximos 10 años, el mercado de los SE crecería de forma exponencial [Graaf, Lormans & Toetenel, 2003]. Ahora bien, de cumplirse este pronóstico, el volumen estimado de ventas del mercado mundial de estos sistemas sería de 194 billones de euros para el 2010 [Arilla & Arribas, 2009]. Esto ha originado que se busque mejorar la calidad y la productividad de los SE.

Para el desarrollo de este tipo de sistemas se ha requerido de herramientas y métodos especializados para obtener sistemas robustos [Vahid & Givargis, 2002] [Berger, 2002]; que cumplan con los requisitos de tamaño, fiabilidad, consumo y costo. Esto da cabida a toda una serie de técnicas, herramientas y metodologías de diseño tanto hardware como software que han impulsado una serie de iniciativas de mejora.

Está claro que el proceso de desarrollo de los SE puede estar lleno de retos: requisitos poco especificados, plazos de entrega ajustados, disponibilidad tardía del hardware, y la falta de visibilidad de la interacción de los sistemas; problemas que en conjunto acaban por hacer todavía más complejo el proceso de desarrollo. Todo esto, aunado al incremento de la complejidad y a la cantidad y variedad del software disponible para estos productos, crea un gran desafío para el desarrollo de este tipo de sistemas. Esto queda evidenciado en los resultados obtenidos en el estudio "*Embedded Market Survey*" del 2008 [URL-2] (véase Figura 1.1).

Los resultados muestran tres principales preocupaciones para los desarrolladores de SE: el calendario de reuniones, el proceso de depuración, y el aumento de líneas y complejidad del código. Es importante hacer notar que la principal preocupación de los desarrolladores es el calendario de reuniones; ya que éste se relaciona directamente con el método de desarrollo de software, específicamente con la etapa de planificación. De acuerdo a [URL-2], el 59% de los encuestados indicó que no utilizan ningún método de desarrollo formal o técnica para llevar a cabo los proyectos empotrados.

En este sentido, de acuerdo a [Greene, 2004] los proyectos carecen a menudo de una metodología de desarrollo eficaz, lo que ha ocasionado que muchas empresas hayan mejorado las metodologías de desarrollo de software [Sangiovanni & Martin, 2001].

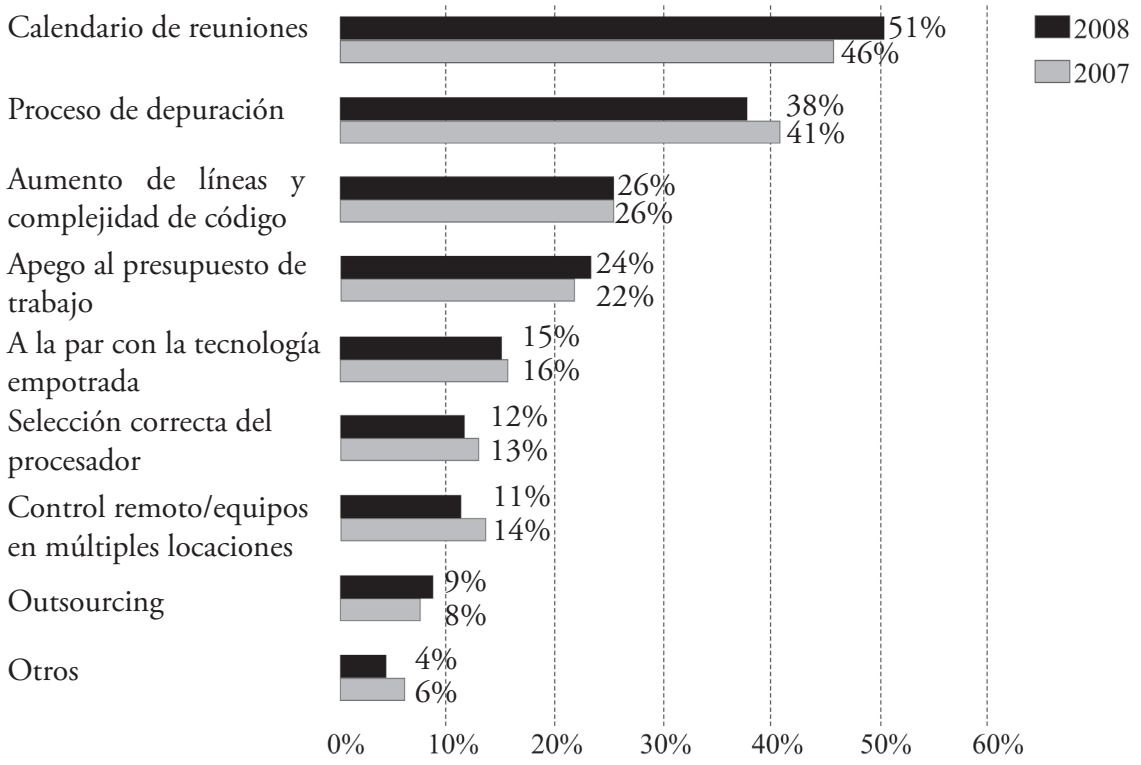


Figura 1.1. Resultados del estudio "Embedded Market Survey" del 2008 [URL-2]

1.2. JUSTIFICACIÓN

En la actualidad los hogares promedio pueden tener más de 50 SE [Srinivasan, Dobrin & Lundqvist, 2009]; que desde una perspectiva empresarial ofrecen una gran cantidad de oportunidades para el mercado mundial. El desarrollo de estos sistemas se ha convertido en un área estratégica para muchas empresas que buscan aumentar su competitividad a través de optimización de la productividad, la calidad, y la puntualidad del desarrollo de SE mediante la adaptación de las tecnologías de Ingeniería de Software que son apropiadas para situaciones específicas.

Sin embargo, las tecnologías disponibles para el desarrollo de software no tienen en cuenta las necesidades específicas de desarrollo de SE, dejando de lado la parte del hardware o no prestándole la atención adecuada a la integración del hardware y del software.

1.2.1. NECESIDAD DE UNA METODOLOGÍA DE DESARROLLO PARA SE

El desarrollo de SE no es una tarea fácil. Es común encontrar sistemas que no satisfacen las necesidades de los clientes y que incurren en el aumento de los costos y en el incumplimiento de los plazos establecidos para su desarrollo. Por ejemplo, el 60% de los componentes que integran tanto el hardware y software deben ser rediseñados luego de haber sido programados [González & Urrego, 2008].

Entre los principales hechos que causan problemas para el desarrollo de los SE se encuentran los siguientes:

1. Por lo regular, la construcción de estos sistemas se realiza por expertos en electrónica, que en general desconocen el uso de las metodologías de análisis y diseño de sistemas [González & Urrego, 2008].
2. La ausencia de metodologías especificas, que son reemplazadas por aquellas de propósito general que no contienen ninguna adaptación a los SE y que no cubren todas las fases de desarrollo [González & Urrego, 2008].
3. Los métodos utilizados, las herramientas y técnicas para el desarrollo de SE varían entre empresas, institutos de investigación y universidades. Lo anterior se debe a la gran aplicabilidad de estos sistemas, y a la ausencia de normas o estándares.
4. Los SE muchas veces no presentan una clara distinción entre hardware y software. El software es en muchos casos una extensión del hardware, por lo que los componentes del hardware son reemplazados por algoritmos controlados por software.
5. El software empotrado es uno de los segmentos más dinámicos y en constante cambio de la industria electrónica que durante los últimos 20 años, ha impactado en la funcionalidad de los SE.

Por todo lo anterior resulta de vital importancia trabajar en el desarrollo de metodologías de diseño capaz de abordar la complejidad de los SE dentro de las pequeñas ventanas de tiempo que permite la dinámica empresarial y las restricciones de mercado. Por lo que, se identificó la necesidad de una metodología que abarque los aspectos tanto de software como de hardware, que facilite el desarrollo de los SE, organice los tiempos, reduzca los riesgo de implementación y el costo, mejore el desempeño y aumente la calidad del producto. Así mismo, asumiendo que el desarrollo de los SE es un motor clave de la industria, es necesario estandarizar los métodos, herramientas y técnicas usadas; y pasar de lo genérico a lo especifico, para aumentar la productividad y la calidad del producto.

1.2.2. BENEFICIOS DE UNA METODOLOGÍA PARA LA MEJORA DEL PROCESO DE DISEÑO DE SE

La metodología propuesta para la mejora del proceso de diseño de SE, se enfoca en el análisis, especificación, y diseño de cada fase del proceso y pretende cubrir los siguientes aspectos:

1. Definición de las fases de los procesos, actividades y subactividades; indicando qué usar, cómo usarlo y cuándo usarlo.
2. Definición de las especificaciones técnicas y de las directrices que definirán la forma de realizar y adaptar estas especificaciones y las fases.
3. Establecimiento de las tareas para el equipo hardware y el equipo software, especificando el conjunto de actividades a realizar para guiar a los desarrolladores durante todo el ciclo de diseño.
4. Recomendación de las herramientas, métodos y técnicas para cada actividad y definición de plantillas que ilustrarán los formatos de los diferentes documentos, y cómo cumplirlas.
5. Especificación de los elementos y de las relaciones entre los componentes del sistema.
6. Diseño de sistemas robustos, capaces y seguros.
7. Optimización del tiempo de desarrollo.

1.3. DELIMITACIONES DEL TRABAJO

- La metodología producto de esta tesis se centró específicamente en el diseño de SE hasta su fase de prototipo. Esto permite asegurar que la metodología desarrollada cumple con su objetivo inicial.
- El desarrollo de esta tesis se enfoca en el Nivel 2 de madurez del CMMI-DEV, por lo que la metodología de diseño de SE cubre los problemas relacionados con las áreas de proceso de este nivel intentando eliminar los problemas mencionados en los primeros apartados de este documento.

1.4. LIMITACIONES DEL TRABAJO

- La metodología propuesta fue validada en un entorno universitario por lo que en un inicio los resultados aquí mostrados no podrán ser reproducidos en un entorno industrial hasta que se obtengan resultados que demuestren su aplicabilidad.

1.5. HIPÓTESIS

La creciente necesidad de estándares y metodologías específicas para el desarrollo de SE plantea la siguiente hipótesis para este trabajo de tesis:

Es posible desarrollar una metodología para el diseño de Sistemas Empotrados bajo el paradigma de la Mejora del Proceso Software; que planifique, administre y controle el desarrollo de estos sistemas a través de la definición de las fases de los procesos, actividades y subactividades; indicando qué usar, cómo usarlo y cuándo usarlo. La metodología tendrá el objetivo de mejorar la calidad del producto final y la productividad en el desarrollo de los Sistemas Empotrados.

1.6. OBJETIVOS DEL TRABAJO

El objetivo general propuesto fue el siguiente:

Desarrollar una Metodología para el Diseño de Sistemas Empotrados bajo el paradigma de Mejora del Proceso Software.

Para alcanzar este objetivo general fue necesario cumplir los siguientes objetivos particulares:

1. Realizar una revisión de las fuentes de información relacionadas con el tema de investigación para establecer el contexto actual del tema: “Metodologías para el desarrollo de SE” y así, identificar la problemática actual.
2. Diseñar una solución con base en el avance científico realizado por la comunidad internacional y a los conocimientos adquiridos en la Maestría en Electrónica y Computación.
3. Establecer la estructura de la solución diseñada; describir elementos, roles, documentos, actividades y componentes de la solución planteada.
4. Mantener la conformidad con las normas existentes para avanzar en la estandarización de SE; y agilizar así el proceso de desarrollo.

5. Desarrollar por lo menos dos proyectos piloto para probar la solución y recoger los resultados.
6. Contar con experiencia para madurar los procesos implicados en el desarrollo de SE a través de una base de conocimiento.
7. Recoger la experiencia para madurar los procesos involucrados en el desarrollo de SE a través de una base de conocimiento.

1.7. APROXIMACIÓN A LA SOLUCIÓN

Software Process Improvement for Embedded Systems (SPIES), es la metodología que se propone para la mejora del proceso de desarrollo de SE y que fue producto de esta tesis. La Figura 1.2 muestra los elementos que interactúan para establecer la metodología para el diseño de SE y que se enfoca a mejorar el proceso de desarrollo completo.

Figura 1.2. Actividades de SPIES

Como se muestra en la Figura 1.2, la metodología SPIES utiliza un repositorio de artefactos para introducir la mejora de los procesos en cada fase. Este repositorio almacena las plantillas para cada actividad, las directrices para llevar a cabo las actividades, la asignación de roles, entre otras cosas.

Las actividades de la metodología SPIES se adaptan de las áreas de proceso y prácticas específicas del CMMI-DEV v1.2 Nivel 2, las cuales fueron simplificadas para las demandas de los SE. La Tabla 1.1 muestra las áreas de proceso del CMMI-DEV v1.2 Nivel 2 que la metodología SPIES cubre. De la misma forma que CMMI-DEV v1.2 lo hace para el desarrollo de software, SPIES recomienda para cada actividad la utilización de una herramienta específica para la planificación del proyecto, el modelado de los requisitos, la validación del sistema, y demás.

SPIES fue diseñada para organizarse por fases que se componen de actividades más específicas (véase Figura 1.3). A través de ocho fases la información fluye en forma de procesos activos. El repositorio de conocimientos gestiona toda la información sobre el proyecto para mejorar continuamente el proceso de diseño de SE. El enfoque iterativo de SPIES asegura que el proceso de desarrollo es analizado en cada fase y no hasta el final del proyecto.

Ahora bien, SPIES se diseñó de forma que pueda utilizarse a través de tres capas dependientes y progresivas:

Tabla 1.1. Áreas de proceso para CMMI-DEV v1.2 Nivel 2 [CMMI, 2006]

Área de proceso	Categoría	Nivel de madurez	Nivel de capacidad 1	2	3	4	5
Gestión de requisitos	Ingeniería	2	■	■			
Planificación de proyectos	Gestión de proyectos	2	■	■			
Supervisión y control del proyecto	Gestión de proyectos	2	■	■			
Acuerdo de Gestión de Proveedores	Gestión de proyectos	2	■	■			
Medición y análisis	Soporte	2	■	■			

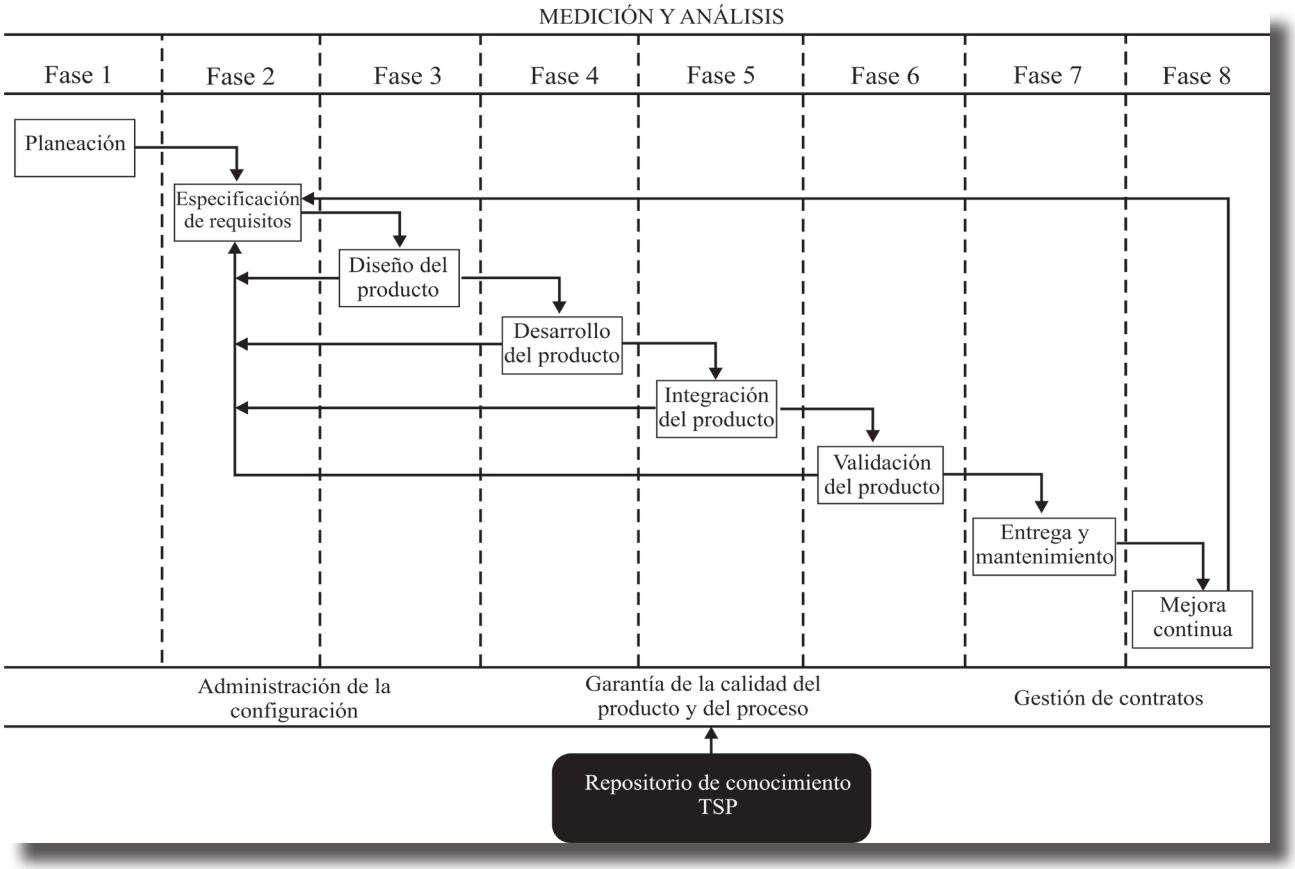


Figura 1.3. Fases y procesos de SPIES

1. **Capa de gestión.** Establece el proceso necesario para controlar el desarrollo del proyecto conjunto. Se inicia con la realización de un plan de proyecto y termina con las lecciones aprendidas en la etapa final. Estas lecciones deben ser almacenadas en el repositorio de conocimiento en forma de esfuerzos necesarios, tiempo, recursos, personas y demás. Esta capa esta compuesta por dos áreas de proceso: Planeación (EPL) y Mejora Continua de Productos (PCI).

2. **Capa de desarrollo.** Establece las actividades relacionadas con el desarrollo del sistema completo. Hasta donde fue posible, los artefactos de *Team Software Process* (TSP) [Humphrey, 2000] se adaptaron para utilizarse en cada fase de SPIES (incluido el plan de proyecto en la capa anterior). SPIES determina cuando los desarrolladores puedan comenzar la próxima etapa a través de fases y criterios de salida. La capa esta compuesta de seis áreas de proceso: Especificación de Requisitos (RES), Diseño del Producto (PDS), Desarrollo del Producto (PDE), Integración del Producto (PIN), Validación (PVAL), y Entrega y Mantenimiento del Producto (PDM).

3. **Capa de soporte.** Proporciona la ayuda para alcanzar la calidad esperada por el establecimiento de actividades para la configuración y la dirección de contratos y medición continua de ellos. La capa se compone de tres áreas de procesos: Gestión de la Configuración (CMA), Productos y Procesos de Calidad (PPQ), Gestión de Contratos (CMG) y Medición y Análisis (MAN).

1.8. ESTRUCTURA DE LA TESIS

La estructura del documento de tesis se basa en siete capítulos; que son detallados a continuación.

Capítulo 1. Introducción. En este capítulo se explica de forma concisa el tema a desarrollar, por lo que se presentará en forma abreviada lo que se desea lograr con la tesis, su justificación y la forma en que se abordará el problema.

Capítulo 2. Revisión de la literatura sobre SE. En este capítulo se presentan los antecedentes teóricos y las líneas actuales de investigación acerca de las metodologías, herramientas y técnicas de desarrollo, que se han realizado sobre el ámbito de los SE.

Capítulo 3. Mejora al Proceso Software. En este capítulo se explica el por qué la producción de software se debe convertir en un proceso disciplinado y aceptado por todos los involucrados en el desarrollo.

Capítulo 4. Introducción a la solución. En este capítulo se definen teóricamente los procesos que la metodología utiliza. Además de que se define también la funcionalidad de las plantillas, actividades, y demás componentes de la solución propuesta.

Capítulo 5. Metodología SPIES. En este capítulo se describe explícitamente la metodología desarrollada.

Capítulo 6. Resultados y discusión. En este capítulo se demuestra la aplicabilidad de la metodología desarrollada a través de su validación en entornos reales. Cabe recordar que la validación se realizó a nivel académico con alumnos de la Maestría en Electrónica y Computación de semestres avanzados. Por último, se presentan los resultados obtenidos y se establece una discusión alrededor de los SE diseñados.

Capítulo 7. Conclusiones y trabajo futuro. En este capítulo se enuncian los principales hallazgos obtenidos durante el proceso del desarrollo de la tesis, así como también se exponen nuevas interrogantes o problemas, que se dejen sin solución.

Al final de la tesis, se presentan los Apéndices a los que se hace referencia a lo largo del documento y por último se presenta la bibliografía utilizada como base para el desarrollo de la investigación.

1.9. PUBLICACIONES GENERADAS

A continuación se enlistan algunas de las publicaciones que se generaron durante el desarrollo del presente trabajo.

Autores:	García, I. & Herrera, A.
Título:	"Using the Software Process Improvement approach for defining a Methodology for Embedded Systems Development using the CMMI-DEV v1.2"
Congreso/Revista:	The 10th International Conference on Computer and Information Technology. Bradford, UK. June 29- July 01, 2010. pp. 233-240.
Publicación:	IEEE Computer Society
Estado actual:	Publicado
Año:	2010

Autores:	García, I. Pacheco, C. & Herrera, A.
Título:	"Combining the Software Process Improvement approach for Embedded Systems Design using the CMMI-DEV v 1.2 model"
Congreso/Revista:	Advances in Computer Science and Engineering, vol 45, pp. 127-144. ISSN: 1870-4069.
Publicación:	Centro de Investigación en Computación, Instituto Politécnico Nacional.
Estado actual:	Publicado
Año:	2010

Autores:	García, I., Pacheco, C. & Herrera, A.
Título:	"Defining a Software Process Improvement-based Methodology for Embedded Systems Development"
Congreso/Revista:	2010 IEEE Electronics, Robotics and Automative Mechanics Conference. CERMA 2010. Cuernvaca, Morelos, México. 28 Septiembre - 1 Octubre 2010. ISBN 978-0-7695-4204-1. pp. 120-125.
Publicación:	IEEE Computer Society
Estado actual:	Publicado
Año:	2010

Autores:	Iván García Pacheco, Carla Pacheco Agüero, Andrea Herrera Huerta
Título:	"Paradigma de mejora al proceso software para el diseño de Sistemas Empotrados"
Congreso/Revista:	VI Semana Nacional de Ingeniería Electrónica. SENIE 10. Huajuapan de León, Oaxaca, México. 13, 14 y 15 de Octubre 2010. ISBN: 978-607-477-363-7. pp. 637-647.
Publicación:	Universidad Autónoma Metropolitana-Azcapotzalco
Estado actual:	Publicado
Año:	2010

Autores:	García, I. & Herrera, A.
Título:	"Desarrollo de una Metodología para el Diseño de Sistemas Empotrados bajo el Paradigma de Mejora del Proceso Software"
Congreso/Revista:	Presentación de poster en la 11a Feria de Posgrados de Calidad 2010. México, Mayo 17-18, 2010.
Año:	2010
Organismo:	CONACYT

2

EL CONTEXTO DE LOS SISTEMAS EMPOTRADOS Y LA CALIDAD DEL PRODUCTO

El desarrollo de dispositivos, equipos y sistemas electrónicos ha ido experimentando avances sucesivos desde que, hace décadas, surgieran los primeros componentes discretos. Sin embargo, las tareas que les son asignadas en la actualidad aumentan su complejidad, la cual sólo puede ser asumida por componentes altamente integrados cuya evolución más reciente es el SE [Salgado, 2008].

En este capítulo se abordarán los conceptos pertinentes a los SE, su historia, su clasificación y los retos actuales en cuanto a su desarrollo.

2.1 DEFINICIÓN FORMAL DE SISTEMA EMPOTRADO

Existen una gran variedad de definiciones para los SE, que son correctas pero subjetivas. A continuación se presentan algunas de estas:

1. Graaf, Lormans, y Toetenel definen a los SE como "*una mezcla de hardware y software que está dedicada a una aplicación especifica, y que forma parte de un sistema físico mayor unido por, al menos, una conexión lógica*" [Graaf, Lormans & Toetenel, 2002].
2. De acuerdo a Galeano [Galeano, 2009] un SE es "*un circuito electrónico computarizado que está diseñado para cumplir una labor especifica de un producto*". Así, la palabra empotrado ha reflejado el hecho de que estos sistemas se incorporan a un sistema de Ingeniería más general [Li & Yao, 2003], en el que se han realizado funciones de control, procesamiento y/o monitorización [Berger, 2002].

A partir de las definiciones anteriores y de la investigación realizada, se define a los SE de la siguiente manera:

3. Un SE es una combinación de software y hardware y algunas otras partes, mecánicas o de otro tipo, destinadas a desempeñar una función específica; cuyo objetivo es optimizar el producto final reduciendo su tamaño o costo, o mejorando características relacionadas con su funcionalidad (eficiencia, disponibilidad, etc.). Estos sistemas se incrustan en un producto más grande y normalmente no son visibles para el usuario. Además, el software empotrado es vital pues los convierte en sistemas de procesamiento informático.

Con base en estas definiciones se puede afirmar que el desarrollo de SE comprende tanto la parte hardware como software, e implica un diseño paralelo de hardware y software.

El software se vuelve un factor limitante (tanto en calidad como en tiempo de desarrollo y en costo), y el uso de técnicas rigurosas de desarrollo de software puede contribuir en gran medida a la calidad de los sistemas, así como al tiempo de comercialización de nuevos dispositivos o familias de dispositivos. Mientras que conocer el funcionamiento del hardware es trascendente y es uno de los requisitos fundamentales para realizar un buen diseño ya que, entre otras cosas, el hardware delimitará las capacidades del sistema que pueden mejorarse con el software.

Una decisión de hardware puede afectar al software y viceversa; por lo que es necesario considerar para su diseño y desarrollo dos elementos conceptuales: la arquitectura hardware (construida en base de un procesador y de dispositivos lógicos programables) y la arquitectura software (que involucra a sistemas operativos, lenguajes de programación, compiladores y herramientas de modelado) [Ball, 2002] (véase Figura 2.1).

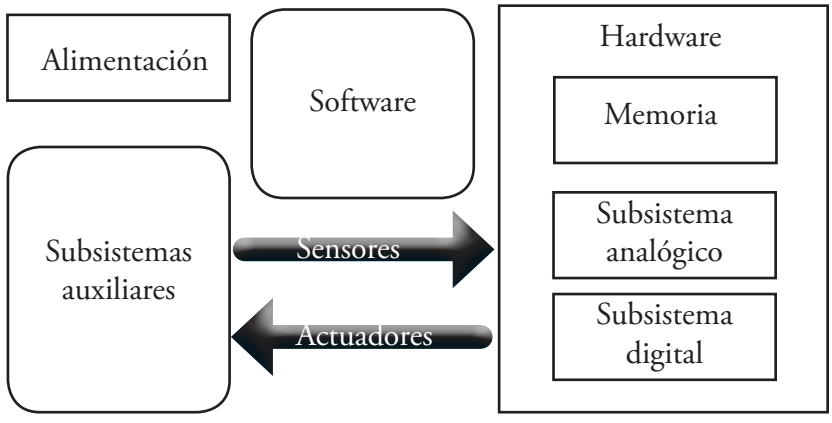


Figura 2.1. Arquitectura típica de un SE

El desarrollo de cada SE es muy específico en cuanto al producto y su aplicación. Sin embargo, el desarrollo de un SE básicamente incluye un ciclo de vida con los siguientes pasos:

- **Especificación del producto:** describe lo que será y lo que hará el producto final [Ball, 2002]; en esta fase se establecen los requisitos y en función a estos se eligen las herramientas de desarrollo hardware y software [Berger, 2002]. Para esta etapa del diseño existen diferentes herramientas para formalizar el SE de acuerdo a las funciones y restricciones que debe satisfacer.
- **División hardware/software:** identifica los subsistemas o módulos que serán implementados vía software o hardware [Berger, 2002]. En esta etapa se debe considerar que los requisitos de hardware son más rigurosos que los requisitos de software; la división depende en gran medida del procesador seleccionado.
- **Diseño de software:** la calidad del software debe ser inherente desde el principio e incorporada mediante el diseño [Calero, 2010]. Para el diseño del software empotrado (conocido también como firmware) existen diversas maneras de describirlo, esto depende de qué información ha de enviarse; algunos de los métodos más utilizados son los diagramas de flujo, diagramas de estado, pseudocódigo, etc. Así, para realizar un buen diseño se debe contar con los requisitos debidamente documentados y contemplar las características del producto.
- **Diseño de hardware:** una vez que el sistema está diseñado y los requisitos de hardware se han establecido, el siguiente paso es diseñar el hardware [Ball, 2002]. El objetivo es tener un diseño detallado del sistema a nivel hardware que cumpla con los requisitos del sistema. Por lo tanto en esta fase se realizan tareas específicas para el desarrollo del hardware (definición de la interfaz hardware, requisitos de tamaño, consumo, etc.).
- **Integración del sistema:** integra los componentes hardware con los componentes software. Una vez que se cuenta con un prototipo a nivel hardware y con el software empotrado compilado sin errores se puede iniciar con esta etapa. Es importante realizar pruebas de hardware y depurar el software simultáneamente usando las herramientas y métodos especiales para el manejo de la complejidad, por ejemplo: Matlab/Simulink, LabView y Proteus.
- **Pruebas del producto:** determinan si el sistema funciona correctamente; estas pruebas son más estrictas. Existen métodos de depuración para SE, sin embargo muchos de estos sistemas no pueden depurarse hasta que se encuentran operando.
- **Mantenimiento y actualizaciones:** la mayoría de los diseñadores de SE (alrededor del 60%) mantienen y mejoran los productos existentes, en lugar de diseñar nuevos productos. La mayoría de estos diseñadores no suelen ser miembros del equipo de diseño original, por lo que se debe confiar en su experiencia, conocimientos técnicos, la documentación existente y el producto; para entender el diseño original y mantenerlo y mejorarlo [Berger, 2002].

Regularmente, los SE son implementados en placas únicas o en un solo chip y deben de comprender tres aspectos: procesamiento, almacenamiento y comunicación [Vahid & Givaigis, 2002]. De acuerdo a [Noergaard, 2005], los sistemas de placas únicas son sistemas en los que se encuentran integrados los recursos de hardware en una tarjeta y son clasificados en cinco categorías de acuerdo a la funcionalidad del componente, estas son: Unidad Central de Procesamiento (CPU), memoria, dispositivos de entrada, dispositivos de salida, y las rutas de conexión (bus de datos). La Figura 2.1 muestra que los SE están conectados con el entorno físico a través de sensores, y su estructura

puede integrarse por FPGAs o partes dedicadas: dispositivos ASIC (Circuitos Integrados de Aplicación Específica), DSP (Procesador Digital de Señales), microcontroladores, etc. [Edwards *et al.*, 1997]. Dado que los SE interactúan continuamente con el medio ambiente que es de naturaleza analógica, normalmente se usan convertidores A/D y D/A, además de componentes electrónicos externos que realizan el resto de tareas necesarias. Los SE implementados en un solo chip cuentan con todos los recursos ya mencionados de forma interna y proporcionan un mayor rendimiento y menor consumo de energía; sin embargo su complejidad es mayor. Para el desarrollo de los SE, es necesario tener en cuenta todos los aspectos de hardware y de software; a continuación se analizarán los componentes más relevantes del hardware.

2.1.1. EL PROCESADOR

De acuerdo a [Marwede, 2006], en el 2006 se estimó que un 79% de todos los procesadores utilizados eran empleados en SE. El procesador (o CPU) es la unidad principal de los SE que ejecuta el software almacenado en la memoria y se encarga de realizar las operaciones como: cálculos matemáticos, ejecución de código para realizar una determinada tarea, además de gestionar el funcionamiento de los subsistemas. Los SE pueden emplear uno o varios procesadores digitales en formato microprocesador, microcontrolador, DSP, DSC, FPGA o usar tarjetas especiales para el desarrollo de SE dependiendo de la complejidad del sistema [Noergaard, 2005].

Inicialmente los SE eran construidos con procesadores de propósito general sin embargo estos procesadores complejos de diseño, incrementaban los costos del sistema debido a sus características y amplio espectro de funcionalidades. Los avances realizados en la tecnología de procesadores en los últimos años, permitieron diseñar e implementar los SE utilizando procesadores especiales en lugar de procesadores de propósito general. Estos procesadores son procesadores de propósito especial diseñados para una clase específica de aplicaciones [Li & Yao, 2003].

De acuerdo a la Figura 2.2, la selección del procesador a usar es una de las tareas principales para el desarrollo de SE, esta tarea se realiza por el staff de Ingeniera de hardware y/o de software considerando los siguientes puntos [Ball, 2002]:

- Número de pines de entrada y salida necesitados.
- Interfaces y memoria requerida.
- Número de interrupciones.
- Consideraciones de tiempo real.
- Entorno de desarrollo.
- Velocidad necesaria de procesamiento.
- Arquitectura de la memoria.
- Requisitos de alimentación.
- Disponibilidad, precio en el mercado y necesidades reales.
- Costos del ciclo de vida.

La oferta en el mercado de semiconductores, tanto de microprocesadores, microcontroladores, DSP y componentes similares, es elevada y se requiere de una cuidada fase de estudio inicial para seleccionar el más adecuado de acuerdo a la aplicación. En la Tabla 2.1 se presenta una lista de los procesadores más comunes para el desarrollo de SE y sus características.

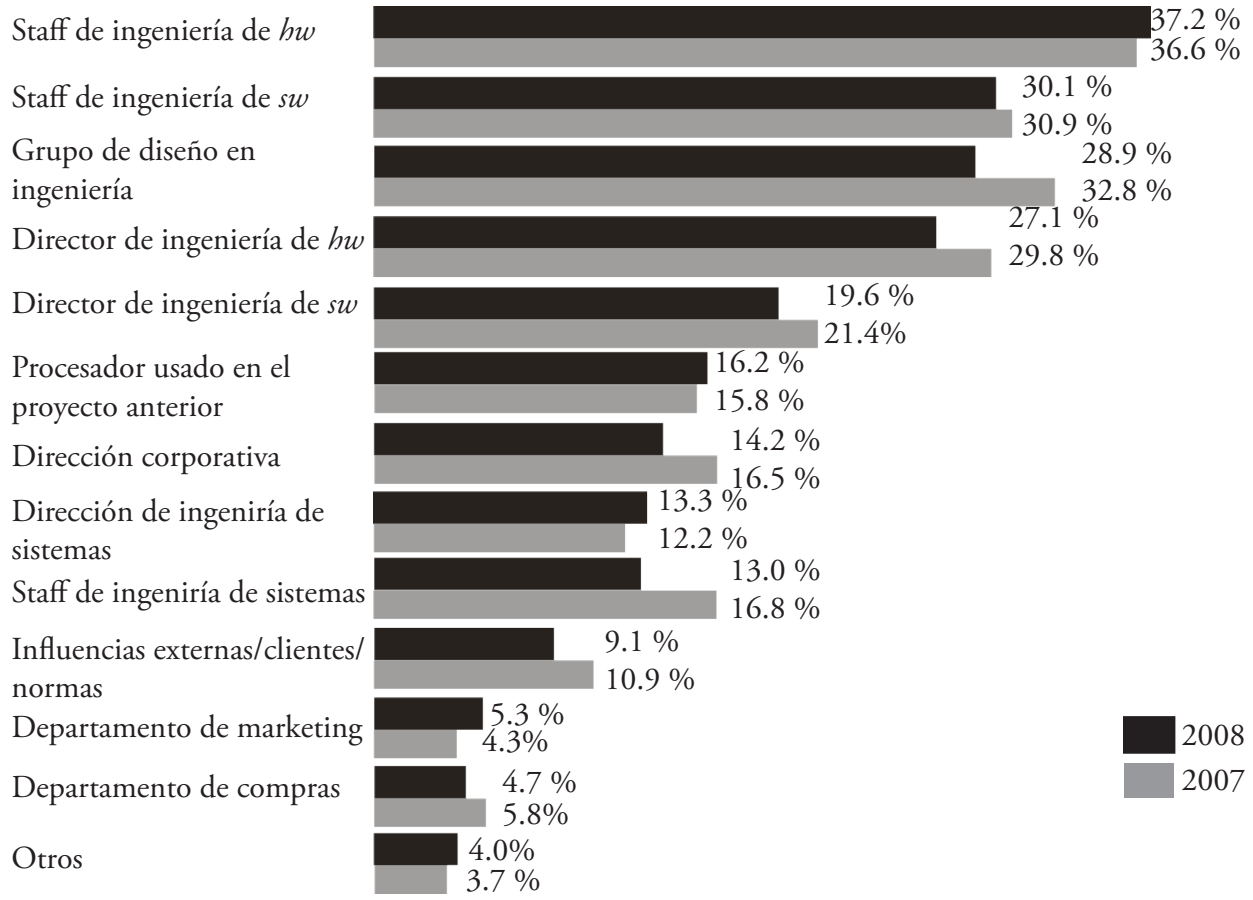


Figura 2.2. Influencias para seleccionar el procesador del SE a desarrollar [URL-2]

Tabla 2.1. Procesadores usados en el desarrollo de sistemas empotrados

Arquitectura	Descripción	Campo de aplicación
ARM	Familia de microprocesadores RISC, cuya principal ventaja es el consumo bajo de energía.	ARM11. Teléfonos celulares inteligentes/ PDA / equipamiento de redes. ARM Cortex-A. Telemedicina/ seguridad / vigilancia/ aviónica.
AMD	Familia de microprocesadores RISC, cuya principal ventaja son sus precios competentes y la tecnología usada.	AMD Phenom 64 Quad, AMD Athlon 64 y AMD Opteron
PowerPC	Desarrollada por IBM, Motorola y Apple. Son procesadores con arquitectura tipo RISC.	PowerPC E600. Redes/ telecomunicaciones.
AVR	Familia de microprocesadores RISC de ATmel.	Tiny AVR ATtiny11/ AVRATmega2560/ ATMEL AT90S1200/ ATmega2560

2.1.1.1. MICROPROCESADOR

Un microprocesador es un controlador complejo síncrono y programable en un solo circuito integrado (CI) que incluye los circuitos de las unidades de control y aritmética-lógica, en otras palabras la CPU, y un conjunto mínimo de memoria integrada y de componentes E/S (I/O) [Tocci & Widmer, 2010]. Para poder realizar su tarea y crear un sistema completo se necesitan otros chips adicionales, tales como memoria, circuitos de entrada salida E/S y reloj; y suelen ser de propósito general.

En 1971, Intel Corporation y el talento creativo de Hoff, Shima, Mazor y Faggin lanzaron el primer microprocesador: el 4004, de 4 bits [Brey, 2002]; lo que marcó una revolución en el campo del diseño de los controladores industriales y de los sistemas lógicos en general, teniendo impacto principalmente en sistemas complejos en lo referente a costo, flexibilidad y minimización de espacio físico ocupado. Desde entonces los fabricantes han realizado muchas mejoras, una de ellas es la expansión de la palabra del microprocesador de 4 a 8, 16, 32 bits (y en algunos casos hasta 64 bits).

Hasta hace unos años los principales fabricantes de microprocesadores eran AMD, Intel y Motorola [Kang, Kwon & Lee, 2005]. Existe una gran variedad de microprocesadores que varían en complejidad, velocidad, tamaño, y capacidad. Sin embargo, su arquitectura básica es la misma. Es común referirse a un microprocesador como la MPU (unidad del microprocesador).

Las unidades elementales de un microprocesador son: unidad de control, unidad aritmética-lógica (núcleo del microprocesador), registros internos, banderas y punteros, sistema de bus de datos, sistema de bus de direcciones y unidades funcionales. El funcionamiento básico de un microprocesador consiste en leer y ejecutar paso a paso todas y cada una de las órdenes (instrucciones) programadas por el diseñador del sistema. Éste se encarga del control y el procesamiento de datos en todo el ordenador. Para esta tarea es necesaria la ayuda de otros elementos capaces de realizar funciones específicas y así liberar de trabajo al microprocesador.

2.1.1.2. MICROCONTROLADORES

La mayoría de los SE han usado microcontroladores en lugar de microprocesadores [Berger, 2002]. El microcontrolador (MCU), es un circuito integrado programable que contiene todos los componentes de un procesador. En su arquitectura interna además de la CPU, están la memoria, los módulos de entrada salida y todos los recursos complementarios (buses, reloj, E/S, otros periféricos tales como conversores A/D, temporizadores, y demás) es decir, se trata de una computadora completa en un solo circuito integrado programable y se destina a gobernar una sola tarea con el programa que reside en su memoria. Sus líneas de E/S soportan la conexión de los sensores y actuadores del dispositivo a controlar. Por lo tanto, es un dispositivo programable capaz de ejecutar instrucciones almacenadas en su memoria de código.

Los microcontroladores están diseñados para proporcionar soluciones de bajo costo, por lo que su uso puede reducir drásticamente los costos de diseño y eliminar tareas redundantes de un proyecto [Craft, 2002]; por eso el tamaño de la CPU, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

Existen varias empresas que se dedican al desarrollo de MCU, algunas son: Microchip, Atmel, FreeScale (antes Motorola), Hitachi, Holtek, Intel, National Semiconductor, NEC, Parallax, Texas Instrument, Zilog, Silab, entre otras. La selección de un MCU depende de la tarea que se quiera realizar y por lo general éste está formado por siete componentes principales: CPU, ROM, RAM, entradas y salidas, temporizadores, buses e interrupciones [Craft, 2002].

2.1.2. MEMORIA

De acuerdo a [Ball, 2002] el determinar los requisitos de memoria también es una parte esencial para el diseño de los SE. En ésta se encuentra almacenado el software que el sistema puede ejecutar así como los datos. En estos sistemas puede existir una jerarquía y diferentes tipos de memoria (RAM o ROM) con diferentes tamaños; cualquier SE tendrá un poco de cada una. Si se requiere sólo una pequeña cantidad de memoria puede estar incluida en el mismo chip que el procesador. De lo contrario, uno o ambos tipos de memoria residirán en chips de memoria externa.

Los SE suelen almacenar todo su código objeto en la memoria ROM, mientras que, la memoria RAM es utilizada para almacenar los datos de entrada o de salida transitoria. La característica principal de la memoria de los SE es que debe tener un acceso de lectura y escritura lo más rápido posible para que el procesador no pierda tiempo en tareas que no son meramente de cálculo; además de considerar la densidad de la memoria. El diseñador de hardware por lo general debe hacer su mejor estimación por adelantado y estar preparado para aumentar o disminuir la cantidad real de memoria que el software utilizará [Barr, 1999]. La cantidad de memoria requerida también puede afectar a la selección del procesador.

2.1.3. ENTRADAS Y SALIDAS

Los SE siguen siendo una colección de piezas programables, y componentes estándar; que interactúan continuamente con un medio ambiente a través de sensores y actuadores [Balarin, *et al*, 1997]. Los sensores y actuadores de un SE son las entradas y salidas del sistema que permiten que el procesador se comunique con el mundo exterior para intercambiar datos. Los sensores proporcionan las entradas de datos hacia los procesadores.

Un sensor es un dispositivo eléctrico y/o mecánico que convierte magnitudes físicas en valores medibles de dicha magnitud. Los sensores van a aportar información tanto del entorno como del estado interno del componente que mide. En los SE suelen utilizarse diversos tipos de sensores como por ejemplo, sensores de luz, sensores de contacto, sensores de temperatura, etc. Las salidas se muestran normalmente como elementos encargados de llevar a cabo las acciones indicadas por el procesador y adoptan la forma de displays y dispositivos electromecánicos (que influyen directamente en el entorno a través de controladores de motores eléctricos, relés, conmutadores, motores eléctricos y otros equipos mecánicos) [Marwedel, 2006]. En esta etapa los SE hacen uso de convertidores digitales a analógicos para convertir la información digital (véase Figura 2.1).

El software determina las tareas que deben realizar los microprocesadores, microcontroladores, DSPs, PLCs, FPGAs u otros dispositivos programables utilizados para el control de equipos electrónicos, eléctricos, electromecánicos, y subsistemas.

El cambio que ha provocado el aumento del software empotrado en productos y servicios ha implicado un cambio riguroso en el desarrollo de los productos empotrados; pues el desarrollo de SE no solo es un desarrollo de hardware con un software añadido. El software empotrado tiene características que lo diferencian de las aplicaciones normales, por ejemplo: la interacción directa con los sistemas hardware, el funcionamiento sin errores y la capacidad de recuperarse por sí mismo en caso de que éstos ocurran y, regularmente, el funcionamiento con recursos limitados.

Un producto empotrado exitoso empieza con un diseño de sistemas flexible, el cual requiere de potentes herramientas de software y un entorno intuitivo de diseño. Es por esto que actualmente el software se ha convertido en el engranaje central de los SE, sin embargo, aunado a las ventajas que ofrece el software empotrado, este debe desarrollarse en el menor tiempo y con calidad para atender los retos de los SE. Así el software empotrado cuenta con diferentes retos entre los que se han encontrado: la creciente complejidad del software, la necesidad de satisfacer las exigencias de alto rendimiento de la convergencia tecnológica, entre otros [Graaf, Lormans & Toetenel, 2003].

El desarrollo de SE inicia con el desarrollo de hardware (que regularmente es necesario modificar hasta obtener el producto final) para después proceder con el desarrollo del software. Es importante considerar que las características que son construidas exclusivamente en hardware tienen costos demasiados altos, pero con el software esto puede cambiar. Las características adicionales que pueden ser implementadas utilizando software no provocan un aumento en los costos de producción comparado con la magnitud en que lo hace el hardware. Por otra parte, los problemas de hardware o defectos de hardware se pueden resolver utilizando las soluciones de software.

Así, las fases básicas para el desarrollo de software empotrado han sido las siguientes: diseño de la arquitectura software, definición de subsistemas software, diseño de bloques funcionales, e implementación de características y tareas [Kang, Kwon & Lee, 2005]. Sin embargo, debido a la importancia que ha adquirido la gestión de la calidad en los SE, el desarrollo de SE ha cambiado a un Desarrollo Guiado por Modelo (*Model-Driven Development,* MDD) y la tendencia de desarrollo considera las siguientes fases: requisitos, diseño funcional, arquitectura del sistema, arquitectura de software, diseño de software, implementación, pruebas de unidades especificas, integración del software y del sistema y pruebas, pruebas funcionales, y validación [Liggesmeyer & Trapp, 2009].

Un denominador común en todos los dominios de software empotrado es el uso de lenguajes de programación que permiten el acceso directo a las interfaces, la memoria, entre otras cosas. Más del 80% de todas las empresas están usando C (y C++), así como tambien, más del 40% están usando el ensamblador para las interfaces de nivel inferior. Mientras que Java se utiliza cada vez para el GUI y programación de aplicaciones.

Otros lenguajes utilizados para desarrollar SE son de alto nivel, por ejemplo: Microsoft .NET Micro Framework, Microsoft .NET Compact Framework o Java Platform Micro Edition, NesC. Actualmente se desarrollan sistemas operativos y herramientas de desarrollo para SE, como lo son: Microsoft (Windows CE, Windows XP Embedded), Wind River (VxWorks, Linux), Symbian (SymbianOS), Palm, Green Hills (INTEGRITY) [Kang, Kwon, & Lee, 2005], Intel® Embedded Design Center, e Intel C++ para Microsoft Embedded Visual C++, Android, Maemo, Network Embedded Systems C, Windows Mobile, eCos, LynxOS, QNX y Linux Embebido (véase Tabla 2.2). Cada proveedor sigue enfoques diferentes dependiendo de su experiencia y posicionamiento en el mercado.

Así, cada vez más la calidad del software está tomando mayor importancia en las organizaciones por su influencia en los costos finales y por ser un elemento diferenciador de la competencia frente a sus clientes.

Tabla 2.2. Sistemas Operativos usados para Sistemas Empotrados

Sistema	Descripción
Android	Android es un stack de software orientado a dispositivos móviles que incluye un sistema operativo, middleware y aplicaciones clave. Está basado en una versión modificada del núcleo Linux, por lo tanto Android es código libre.
ARM *Linux Mobile Platform*	ARM y otras empresas como Texas Instruments, Samsung, Marvell o Mozilla están creando una plataforma basada en Linux especialmente diseñada para dispositivos móviles.
Maemo	Maemo es una plataforma de desarrollo para dispositivos portátiles basado en Linux. Maemo cuenta con una interfaz de usuario optimizada para dispositivos móviles. La mayoría de los componentes Maemo son de código abierto, lo que ofrece a los usuarios y desarrolladores la libertad y flexibilidad para contribuir y modificar el desarrollo de la plataforma central.
OpenMoko	Proyecto para crear una plataforma para smartphones utilizando software libre. Utiliza el núcleo de Linux y está basado en el framework de OpenEmbedded.
QNX	QNX es un sistema operativo de tiempo real basado en Unix. QNX brinda una fuente fiable para aplicaciones en telecomunicaciones, sistemas electrónicos, sistemas automotores, instrumentación médica, control industrial, y más.
Symbian (SymbianOS)	Symbian es un sistema operativo desarrollado por Symbian Ltd. para adaptarse a los requerimientos de un teléfono móvil. Fue producto de la alianza de varias empresas de telefonía móvil (Nokia, Sony Ericsson, Siemens, Lenovo, Motorola, Panasonic, etc.).
Ubuntu Mobile and Embedded	Iniciativa de Ubuntu para crear una versión para MID (*Mobile Internet Devices*); con el objetivo de hacer portable Ubuntu en smartphones. Los programadores Ubuntu desarrollan una versión móvil de su sistema operativo en colaboración con Intel. Una de las ideas principales del proyecto es contar con un procesador de bajo consumo y una arquitectura chipset diseñada para permitir capacidades de Internet completas sobre dispositivos móviles.
Windows Embedded Devices (Windows CE, Windows XP Embedded)	Sistema operativo de Microsoft para dispositivos empotrados. Permite generar una instalación de Windows XP (o CE) adaptada al hardware especifico del dispositivo. Es común encontrarlo empotrado en Sistemas de Posicionamiento Global (GPS), cajeros automáticos, automóviles, dispositivos portátiles, servidores, y demás.

2.2 CLASIFICACIÓN Y CARACTERÍSTICAS DE LOS SISTEMAS EMPOTRADOS

De acuerdo a [Vahid & Givaigis, 2002] los SE pueden clasificarse en tres grupos dependiendo de su interacción con su entorno:

- **Sistemas reactivos:** aquellos sistemas que están en constante interacción con su entorno y se ejecutan a un ritmo determinado por éste. La velocidad de operación del sistema deberá ser la velocidad del entorno exterior.
- **Sistemas interactivos:** aquellos sistemas que siempre interactúan con el exterior de tal forma que la velocidad de operación del sistema deberá ser la velocidad del propio SE.
- **Sistemas transformables:** aquellos sistemas que no interactúan con el exterior, únicamente toman un bloque de datos de entrada y los transforman en un bloque de datos de salida.

Desarrollar un SE consiste en construir una implementación que satisfaga la necesidad deseada, que optimice métricas de diseño (energía, tamaño de código, eficiencia en el tiempo de ejecución, peso, costo) y que cumpla con ciertas características que los distinguen de otros sistemas de cómputo. A lo largo del tiempo, se ha exigido que el diseño de un SE cumpla con las siguientes características:

- **Concurrencia:** los componentes del sistema deben de funcionar de manera simultánea.
- **Fiabilidad y seguridad:** no debe presentar fallos y debe manejar los errores. El manejo de errores se puede hacer a través de hardware o de software [Vahid & Givaigis, 2002].
- **Capacidad de reacción y tiempo real:** ciertos componentes del SE deben reaccionar continuamente a los cambios en el entorno del sistema y deben computar ciertos resultados en tiempo real [Berger, 2002].
- **Eficientes y fáciles de mantener:** el SE debe permitir su modificación después de su lanzamiento inicial.
- **Interacción con dispositivos físicos:** los SE interaccionan con su entorno mediante diversos tipos de dispositivos físicos.
- **Robustez:** los SE deben operar bajo condiciones ambientales extremas [Berger, 2002].
- **Bajo consumo:** Debido al uso de lógica TTL muchos sistemas están alimentados con baterías o pilas.
- **Precio reducido:** un SE es más limitado en funcionalidad de hardware y/o software que una computadora personal lo que implica un bajo costo de fabricación [Berger, 2002].
- **Diseño especial:** los SE están dedicados a tareas específicas, por lo que este tipo de diseños combina hardware personalizado con software empotrado [Balarin, *et al*, 1997], lo que los hace únicos.
- **Flexibilidad:** los SE son especialmente sensibles a aspectos mercadotécnicos, y deben ser capaces de evolucionar con el mercado de una manera flexible y en un periodo determinado de tiempo. Sin embargo, a diferencia del software diseñado para computadoras de uso general, el software empotrado generalmente no se puede ejecutar en otros SE, sin modificaciones significativas [Barr, 1999].
- **Sistemas híbridos:** los SE son híbridos en el sentido de que incluyen partes analógicas y digitales.

2.3 EVOLUCIÓN Y VISIÓN GENERAL DE LA SITUACIÓN ACTUAL DE LOS SISTEMAS EMPOTRADOS

En la actualidad, los SE están presentes en el mundo y los hogares promedio tienen más de 50 SE [Srinivasan, Dobrin & Lundqvist, 2009]. Así, el mercado de los SE es uno de los que presentan mayor crecimiento a nivel mundial. Los SE contribuyen en gran medida a mejorar nuestra vida cotidiana y agregan valor a los productos que los contienen, convirtiéndose en una vía importante de innovación, por lo que se les ha identificado como un área de importancia capital en la mayor parte de los países desarrollados. De acuerdo a [Barr, 1999] parecía inevitable que el número de sistemas empotrados siguiera aumentando rápidamente. En el 2003, Graaf aseguraba que el mercado de los SE crecería exponencialmente en los siguientes 10 años [Graaf, Lormans & Toetenel, 2003]. En la Figura 2.3, se observa que la investigación, desarrollo y uso de los SE ha aumentado con el paso de los años; según las previsiones de ARTEMIS, habrá más de 16,000 millones de dispositivos empotrados en el 2010 y se superarán los 40,000 millones para 2020.

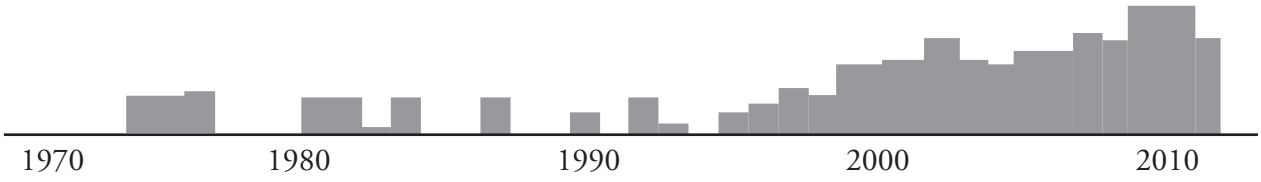


Figura 2.3 Historia de los SE [Gráfica generada con Google]

Los SE pueden ser de muy diferentes tipos y sus principales áreas de aplicación son: control industrial, vídeo e imagen, electrónica de consumo, sector aeroespacial, sector automotriz, equipos médicos y militares, computadoras y periféricos, comunicaciones de datos y telecomunicaciones (en la Figura 2.4 se presenta la distribución de los SE en la industria). Los SE se utilizan en aplicaciones críticas, sólo a modo de ejemplo, se mencionan algunas de estas:

- **Automotriz:** es posible encontrar SE como sistemas de control en automóviles. Los autos modernos sólo pueden venderse si contienen una cantidad significativa de electrónica: sistemas de control de las bolsas de aire, los sistemas de control del motor, los Sistemas de Frenado Antibloqueo (ABS), el aire acondicionado, Sistema de Posicionamiento Global (GPS), las características de seguridad, sistemas de encendido y control del motor.
- **Aplicaciones militares:** las compañías de SE actualmente deben concentrar sus esfuerzos en desarrollar mayor tecnología para el sector militar, en especial Vehículos Aéreos no-Tripulados (UAV), y todo lo relacionado con comunicación militar, además de otras aplicaciones tales como los DSP y de paquetes.
- **Sistemas médicos:** se encuentran SE en equipos de diálisis, equipos de monitorización fetal y todo tipo de equipos médicos y hospitalarios, incluso en equipos de fabricación de productos médicos y equipos de laboratorio.

Tal y como se mencionó, una de las características de los SE es que tienen que reaccionar en un tiempo adecuado y con el apropiado uso de recursos, por lo que estos sistemas también son omnipresentes en la telefonía móvil o la electrónica de consumo (áreas de gran impacto económico), por ejemplo:

- **Aparatos electrodomésticos:** televisores analógicos y digitales, DVD, VCR, asistentes de datos personales, aparatos de cocina (refrigeradores, tostadores, hornos de microondas), juguetes/juegos, teléfonos fijos, teléfonos celulares, cámaras, GPS.
- **Estaciones de red:** ruteadores, concentradores (HUB), gateways.
- **Control industrial:** robots y sistemas de control (manufactura).

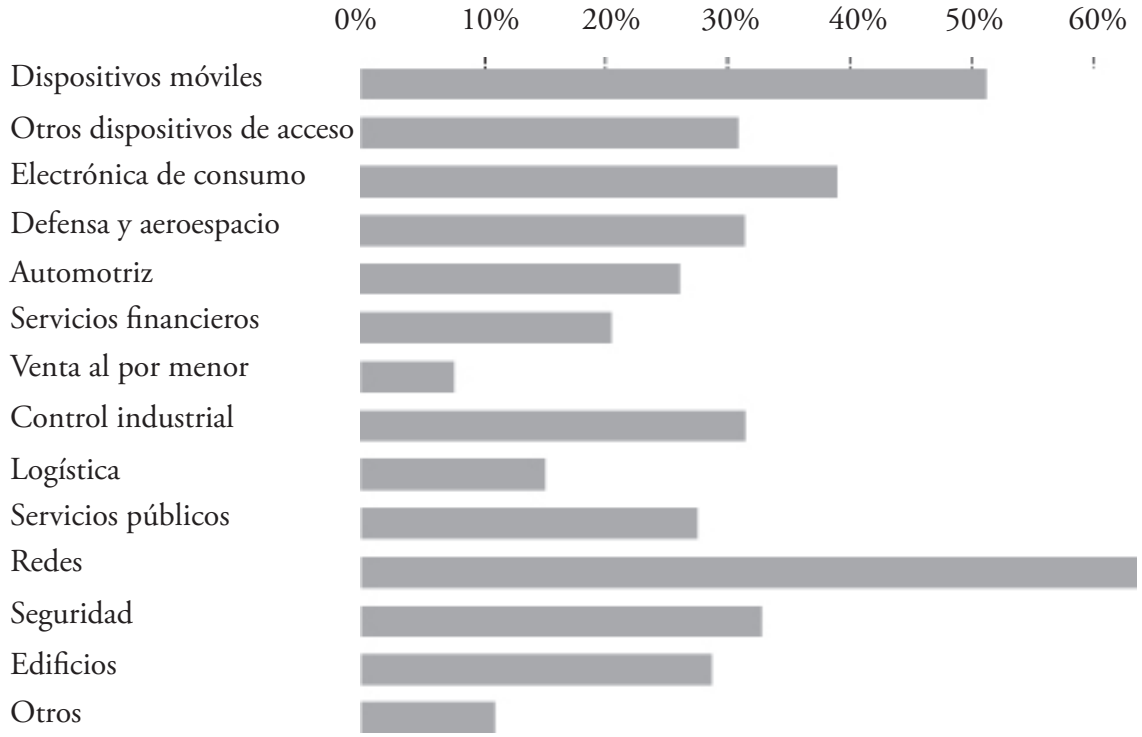


Figura 2.4. Distribución de los SE en la industria [URL-2]

A nivel de diseño de los SE, actualmente existen tres enfoques principales: codiseño hardware/software, plataforma de diseño, y diseño basado en componentes [Shaout, El-Mousa & Mattar, 2010].

- El **codiseño hardware/software** permite optimizar un diseño mediante la partición del sistema en componentes hardware y software (usando un enfoque top-down), considerando factores de calidad específicos. A diferencia del diseño tradicional, se parte de una especificación y se diseña de forma independiente a la arquitectura y posterior implementación del sistema. Sin embargo, surgen comúnmente limitaciones debido a indisponibilidad de herramientas adecuadas para el desarrollo de las distintas etapas del codiseño y a la factibilidad de realizar determinadas implementaciones por razones económicas. De acuerdo a [Srinivasan, Dobrin & Lundqvist, 2009] esta división de desarrollo en dos corrientes independientes, de software y de hardware, ocurre de forma paralela y el resultado de la integración puede convertirse en un punto crítico de fallo.
- ***Diseño basado en plataformas*** introducido desde hace varios años pretendiendo redefinir el futuro de los sistemas en chip (SoC); se ha convertido en un importante estilo de diseño. En general, una plataforma es una abstracción que oculta los detalles de la ejecución de varias posibles mejoras de nivel en capas subyacentes [Sangiovanni & Martin, 2001];

cada plataforma se trata de una colección de elementos. Su uso tiene ventajas y desventajas, pues, este enfoque limita opciones pero reutiliza diseños propiciando que el tiempo de comercialización sea rápido [URL-4]. La reutilización de diseños permite construir fácil y rápidamente SoC con características comunes.

- **Diseño basado en componentes**. El desarrollo de software basado en componentes permite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y un mayor retorno sobre la inversión. Los pasos básicos del diseño basado en componentes son: obtención de requisitos, partición en componentes, diseño interno, evolución del componente, e interconexión de componentes. Entre las ventajas que aporta el uso de este enfoque de diseño se encuentran: reutilización del software, simplificación de pruebas y del mantenimiento del sistema, mayor calidad, ciclos de desarrollo más cortos.

De acuerdo a [Noergaard, 2005], el diseño general de SE define siete fases de desarrollo: especificación del producto, división hardware y software, iteración e implementación, diseño detallado hardware y software, integración de componentes hardware y software, prueba y liberación del producto, mantenimiento y actualización. Esta división de desarrollo en dos corrientes independientes de software y de hardware ocurre de forma paralela, y el resultado de la integración puede convertirse en un punto crítico de fallo [Srinivasan, Dobrin, & and Lundqvist, 2009].

Para describir el ciclo de diseño de los SE se han usado modelos propios de la Ingeniería de Sistemas [Noergaard, 2005]. Estos modelos son:

- El modelo de gran explosión (*Bing-bang model*): en este modelo lo esencial es no planear o procesar los requisitos antes o durante el desarrollo del sistema.
- El modelo codificar-arreglar (*Code-and-fix model*): este modelo no define los procesos formales antes de empezar el desarrollo, pero si los requisitos.
- El modelo de cascada (*Waterfall model*): el modelo de cascada cuenta con procesos definidos para desarrollar el sistema; el resultado de un paso sirve como entrada para el siguiente.
- El modelo de desarrollo evolutivo: se entrelazan las actividades como la especificación, el desarrollo y la validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas y se refina basándose en las peticiones del cliente para producir un sistema que satisfaga los objetivos.
- El modelo de Ingeniera basada en Componentes: se basa en la existencia de un número significativo de componentes reutilizables. El proceso principal de desarrollo se centra en integrar los componentes existentes dentro del sistema más que en desarrollarlos desde cero.

El enfoque actual está introduciendo cada vez más a la Ingeniería de Software, ya que ésta proporciona técnicas y procedimientos que permiten ayudar a asegurar un software de calidad. Así, el desarrollo de SE de calidad bajo el enfoque de la Ingeniería de Software cubre los siguientes aspectos:

- Crecientes posibilidades del hardware.
- Creciente complejidad y tamaño en el software.
- Distribución y composición de equipos de trabajo.

Tal y como se mencionó en el Capítulo 1 de esta tesis, en el 2008 se realizó el estudio llamado "*Embedded Market Study*" que consistió en estudiar las principales áreas de aplicación de los SE; cuyas actividades se centran en la escritura de software, la depuración de software, la integración de hardware/software, elección o especificación de la arquitectura, diseño o el análisis del software, gestión de proyectos, y depuración de hardware.

En los resultados se observó que más de la mitad de los encuestados se encontraban trabajando en una actualización del producto, en lugar de trabajar en un proyecto nuevo. Más de la mitad de estas actualizaciones están aprovechando un microprocesador nuevo, por lo que requiere cambios de software. El tamaño del equipo de diseño promedio aumentó ligeramente, de 13.6 personas a 15.2 personas. Pero es interesante observar que el número de Ingenieros de Software en el equipo aumentó a casi dos, es decir, el número de Ingenieros de Hardware se mantuvo igual o se redujo ligeramente. En relación a lo anterior, la Figura 2.5 muestra el entorno de desarrollo que los desarrolladores de SE están usando. La Figura 2.6 muestra las principales herramientas hardware/software utilizadas para los SE. Es importante resaltar el lugar en el que se encuentran las herramientas de prueba de software, en la parte inferior de la lista. En relación a esto, los usuarios dicen que las pruebas son uno de los factores clave en el proceso de diseño y que toman a menudo la mayor parte del tiempo del mismo. Sin embargo en la lista aparecen en el último lugar, lo que deja como conclusión que los usuarios no están satisfechos con sus herramientas actuales de prueba de software. Es importante observar que las herramientas más usadas son el compilador/ ensamblador, el depurador y el osciloscopio. Por otra parte la necesidad de capturar las especificaciones en niveles altos de abstracción ha llevado al uso de herramientas de modelado como Matlab y Simulink[1] de MathWorks. Estas herramientas permiten a los diseñadores montar rápidamente algoritmos y simular su comportamiento.

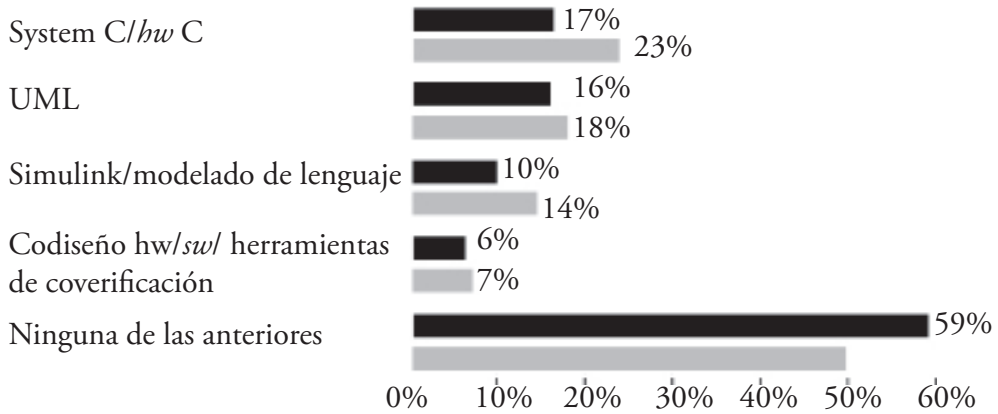


Figura 2.5. Entornos de desarrollo usados para SE [URL-2]

1 Simulink® es una plataforma para simulación y y diseño multidominio en base a modelos de sistemas-dinámicos y empotrados. Proporciona un entorno gráfico interactivo y un conjunto de librerías de bloques personalizables que permiten diseñar, simular, implementar y probar una gran variedad de sistemas con variación temporal, entre los que se incluyen sistemas de comunicaciones, control, procesado de señales, vídeo e imagen.

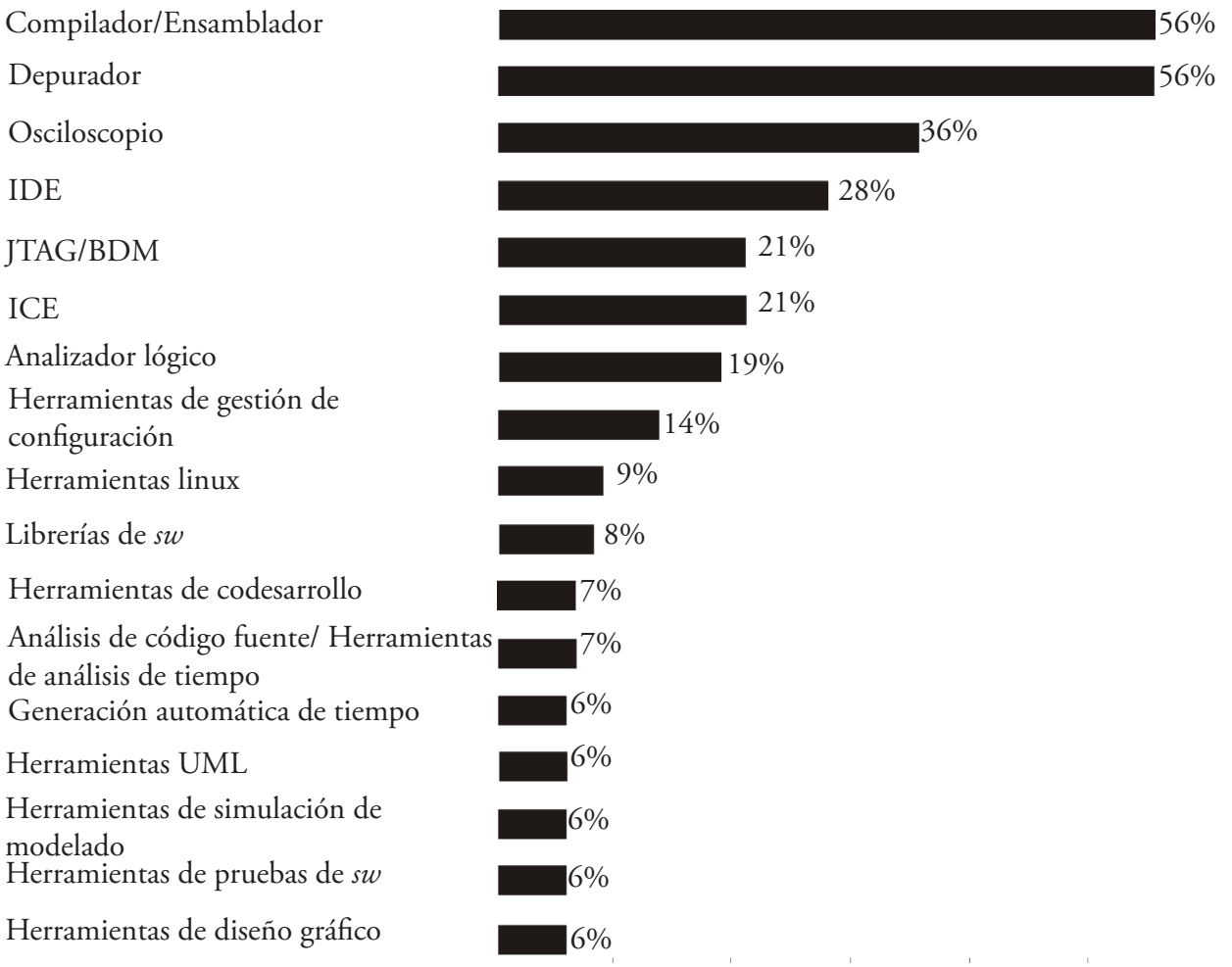


Figura 2.6. Herramientas hardware/software usadas para SE [URL-2]

2.4 RETOS EN EL DISEÑO DE SISTEMAS EMPOTRADOS

El campo de los SE es cada más difícil y las cuestiones de desarrollo de software empotrado están llamando la atención de un número creciente de investigadores, tanto en la industria como en la investigación. Hoy en día, se sigue pensando que los SE están diseñados con un enfoque *ad hoc* que está fuertemente basado en la experiencia anterior con productos similares y en el diseño de manuales, tal y como se determinó hace más de una década en [Balarin, *et al*, 1997].

Los problemas que surgen en el diseño e implementación de estos sistemas involucran prácticamente a todas las ramas de la ingeniería: Computación (tanto en software como en hardware), Comunicaciones, Control, Electrónica y Mecánica. Así, algunos de los principales retos de los SE son:

- **Estandarización:** Durante los últimos años se han utilizado diferentes tipos de metodologías (disponibles para el desarrollo de software que son adaptadas para SE), herramientas y lenguajes; debido a que existen diferentes tipos de enfoques (p.ej. herramientas para ingeniería de requisitos, herramientas de diseño, etc.) [Graaf, Lormans & Toetenel, 2002].

- **Tiempo menor de desarrollo:** antes los SE cumplían con grandes ciclos de vida; en losúltimos años estos sistemas han disminuido constantemente el tiempo destinado para su desarrollo. Las empresas que buscan ser líderes obteniendo una ventaja competitiva de estos productos, realizan una promoción de 6 a 9 meses del SE [Li & Yao, 2003]. Las empresas que no cumplen con estos tiempos para desarrollar o innovar su producto sufren pérdidas en la cantidad de ventas, lo que les ocasiona pérdidas muy importantes respecto a sus expectativas y en el avance tecnológico pues el poder de procesamiento de los microprocesadores sigue aumentando a un ritmo previsto por la Ley de Moore.
- **Bajo consumo de energía y alto rendimiento:** la componente software debe ser capaz de utilizar eficientemente las optimizaciones que la componente hardware pone a disposición de los diseñadores de SE para lograr disminuir el consumo de energía del sistema y aumentar su rendimiento.
- El desarrollo de software está cambiando de la programación manual al Desarrollo Guiado por Modelo.
- IBM *Software Group* afirma que el 80% de los costos de desarrollo se emplean en identificar y solucionar errores.

2.5 PROPUESTAS ACTUALES PARA EL DESARROLLO DE SISTEMAS EMPOTRADOS

En la actualidad se han desarrollado diferentes investigaciones en el dominio de los SE que pretenden gestionar adecuadamente un desarrollo de calidad buscando un equilibro entre los siguientes tres elementos: costo, calidad y tiempo (pues de estos elementos depende directamente el éxito de este tipo de sistemas, tal y como se ha manifestado a lo largo del documento). Los diversos marcos de desarrollo o Entornos de Desarrollo Integrados (*Integrated Development Environment*, IDEs) para SE tienen por objetivo establecer fases para guiar a los desarrolladores en el proceso de desarrollo de SE, así como dar respuesta a los cambios que surgen durante las diferentes fases. A continuación se analizarán las principales propuestas (y de actualidad) con el objetivo de identificar aspectos comunes y establecer así un proceso eficiente de diseño.

2.5.1. *EMBEDDED SOFTWARE COMPONENT MODEL (ESCM)*

De acuerdo a [Li *et al.*, 2009], las tendencias del software empotrado representan una necesidad urgente para la cual es posible emplear técnicas avanzadas para su desarrollo. El desarrollo desde cero es una práctica común, sin embargo los beneficios potenciales del desarrollo basado en componentes; tales como la reducción del tiempo y costo de desarrollo, y el aumento en la calidad, así como la especialización de conocimientos, son muy atractivos en el ámbito de los SE. Pero a pesar de esto, el uso directo de las tecnologías de uso general basadas en componentes para el ámbito de los SE resulta complicado debido a varios retos ya mencionados.

En primer lugar, las aplicaciones empotradas se suelen utilizar en tiempo real y en entornos con alto grado de seguridad. Por lo tanto, las propiedades no funcionales (NFPs), como la fiabilidad en tiempo real y la memoria, tienen que ser modeladas explícitamente para permitir el razonamiento 'composicional' automatizado. Segundo, dado el tamaño pequeño y los requisitos para la movilidad, los SE a menudo suelen estar limitados en recursos.

2.5.1.1. OBJETIVO

Establecer un modelo formal para el desarrollo de software basado en componentes y orientado al campo de aplicación empotrado. El modelo ESCM (*Embedded Software Component Model*) consta de tres elementos esenciales: interfaz, componentes, y conector en combinación con contratos. ESCM se orienta a la especificación, la verificación y la composición de software empotrado y describe cómo especificar los componentes desde cuatro perspectivas: sintáctica, funcional, de calidad del servicio (QoS) y de sincronización. ESCM pone especial atención en los atributos no funcionales de los SE. Así, mediante un contrato llamado "contrato de NFPS" se especifican los atributos no funcionales de este tipo de sistemas.

2.5.1.2. ESTRUCTURA

La Figura 2.7 muestra que ESCM se basa en tres elementos: interfaz, componentes y conector. Cada elemento es especificado a través de uno o varios contratos e información adicional. El marco de ESCM especifica que el contrato de firma, el contrato de funcionalidad y el contrato de los NFPs están relacionados directamente con la interfaz. Mientras que el contrato de interacción es responsable de los componentes; y el contrato de adhesión (contrato "*glue*") es el responsable del conector.

El desarrollo basado en componentes tiene por objetivo desarrollar y mantener sistemas de software a través de componentes ya existentes. Es un sentido común que los componentes deben ser reutilizables y pueden interactuar entre sí en la arquitectura del sistema. A tal efecto, un componente de software empotrado es una unidad de ejecución reutilizable con interfaces especificadas contractualmente y comportamientos coherentes y dinámicos de especificación formal.

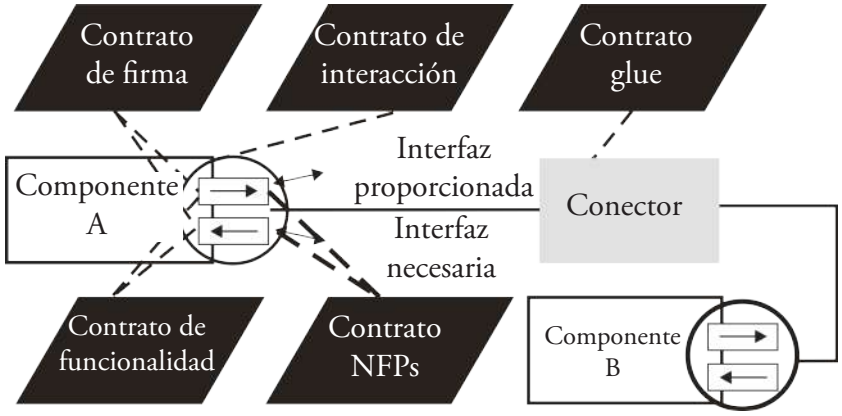


Figura 2.7. Marco de trabajo del modelo ESCM [Li *et al.*, 2009]

Por lo tanto, ESCM define a un componente mediante los siguientes elementos: un conjunto de interfaces (interfaces necesarias e interfaces requeridas) y la descripción del comportamiento dinámico de los componentes. Otro de los componentes del modelo es la interfaz que permite describir la interacción del componente con el resto del sistema de manera abstracta (usando al componente como una caja negra), y determina el comportamiento externo y las características de los componentes.

Así, el marco ESCM define un interfaz como una especificación de servicios expuestos a través de los componentes que interactúan y se puede utilizar en la construcción y mantenimiento de software empotrado. ESCM define una interfaz mediante los siguientes elementos:

- *Tipo*, que describe la interacción entre la interfaz y el entorno del componente,
- *Firma*, que es la especificación a nivel sintáctico que describe cómo usar la interfaz,
- *Funcionalidad*, que es una especificación de nivel semántico de la funcionalidad de la interfaz y presenta una descripción formal a través del contrato de funcionalidad, y
- *NFPs*, muchos modelos se enfocan sólo en los aspectos funcionales de un componente, sin embargo, para el software empotrado no es suficiente y mediante el contrato de NFPs se especifican las limitaciones no funcionales.
- El último elemento del modelo es el conector y este se encarga de proporcionar el «pegamento» para el diseño arquitectónico; es decir, es el elemento encargado de la comunicación y coordinación entre los componentes.

La Figura 2.8 muestra que ESCM distingue cinco tipos de contratos en cuatro niveles. Estos contratos se deben especificar estrictamente para la reutilización de componentes de forma segura. Las etapas para el desarrollo basado en componentes de ESCM son:

- **Especificaciones:** especificar el sistema y los componentes del mismo.
- **Validación:** obtener los componentes propios del repositorio de componentes que coinciden con los componentes especificados. Si no hay componentes existentes reutilizables, entonces se deben desarrollar nuevos componentes según las especificaciones.
- **Composición:** formar el sistema usando el conector y los componentes.
- **Predicción y análisis:** analizar las propiedades del sistema, especialmente los NFP.

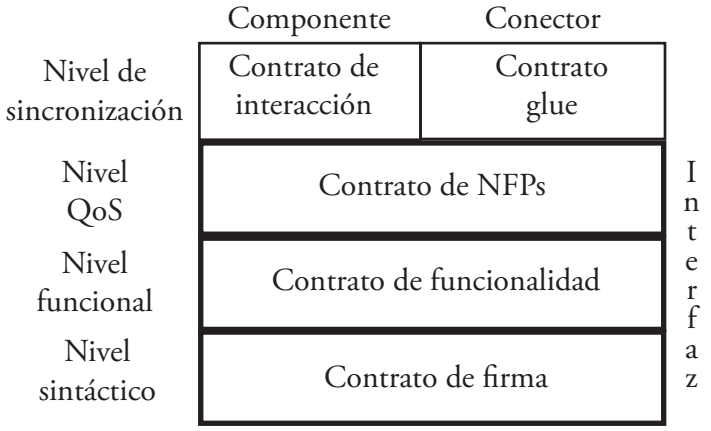


Figura 2.8. Contratos en ESCM [Li *et al.*, 2009]

2.5.2. *SAVE INTEGRATED DEVELOPMENT ENVIRONMENT (SAVE-IDE)*

Algunos dominios de los SE requieren tener una alta confianza en la calidad del desarrollo del producto.

Para ello, un deseo fundamental es tener la capacidad para hacer frente a requisitos tales como la fiabilidad (por ejemplo, disponibilidad y seguridad), la planificación (tales como la libertad y el tiempo de respuesta, tiempo de ejecución, el plazo de desarrollo), y la utilización de recursos (incluyendo la memoria, CPU, canales de mensajes, y/o el consumo de energía). De acuerdo a [Sentilles *et al*., 2009], esto exige un fuerte énfasis en la posibilidad de analizar y automatizar el proceso de desarrollo para garantizar la necesaria calidad de los productos finales con respecto a tales requisitos. Al mismo tiempo, la creciente complejidad de los SE requiere métodos que incrementen el nivel de abstracción, mejoren la reutilización, y permitan la concurrencia en el proceso de desarrollo. Tal y como se dijo anteriormente, un enfoque para conseguirlo es la Ingeniería de Software basada en componentes. Sin embargo, la mayoría de las tecnologías basadas en componentes carecen de las herramientas de análisis formales necesarias para asegurar la confiabilidad. *Save Integrated Development Environment* (Save-IDE), reúne las herramientas y técnicas necesarias para el proceso de desarrollo de SE y los integra con el desarrollo basado en componentes. Esto incluye el desarrollo basado en un modelo de componentes, o SaveCMM, que está diseñado para permitir la eficiencia del diseño de los SE y el análisis temporal del modelo.

2.5.2.1. OBJETIVO

Save-IDE se ha diseñado como una plataforma con un conjunto ampliable de herramientas de apoyo integrado para lograr el enfoque *Save-Comp Component Technology* (SaveCCT). Así, Save-IDE aplica un nuevo enfoque que integra las siguientes actividades: diseño, análisis, transformación, verificación y síntesis.

2.5.2.2. ESTRUCTURA

Save-IDE se basa en el proceso de desarrollo SaveCCT el cual está diseñado como un enfoque top-down con énfasis en la reutilización de componentes. La Figura 2.9 muestra que el modelo comprende tres fases: diseño, análisis, y realización. Si los componentes coinciden con los requisitos que ya existen, se seleccionan y se adaptan a la actividad. De lo contrario, se desarrollan nuevos componentes. En consecuencia, los componentes son analizados y verificados individualmente en base a los requisitos. Después de haber reconstruido el sistema, partes del sistema o las composiciones del sistema deben ser analizadas o verificadas (lo que se denomina como verificación formal del sistema). El sistema y el diseño de componentes y el procedimiento de verificación se repiten hasta que los resultados son aceptables desde el punto de vista del análisis. Enseguida se continúa con la fase de realización, que consiste en la síntesis y la ejecución o simulación de las actividades.

El sistema se sintetiza de forma automática basándose en la entrada del diseño del sistema, en las implementaciones de los componentes y, en algoritmos estáticos para el uso de recursos y las limitaciones de tiempo. El proceso de desarrollo es semi-automático, con varias actividades automatizadas. Una actividad automatizada es la producción del esqueleto de los archivos de aplicación (archivos de cabecera C y sus correspondientes archivos) en base a la especificación del componente. Otra automatización reside en la generación de los de intercambios de archivos utilizados como medio de comunicación entre las herramientas. La tercera se produce durante la síntesis que incluye la transformación de los componentes en las unidades ejecutables en tiempo real, las tareas, la generación de código, la inclusión de un algoritmo de programación

particular, la compilación y la vinculación de todos los elementos en la imagen ejecutable. En la Figura 2.10 se muestra la estructura de Save-IDE, que está compuesta por el editor de la arquitectura donde el sistema y componentes del modelo pueden ser creados. Los componentes individuales pueden ser implementados desde archivos generados en plantillas de C a través de la herramienta del entorno C (CDT de Eclipse). El editor de arquitectura permite realizar la especificación de la interfaz funcional y asignar diferentes atributos a los componentes, tales como el tiempo de ejecución, o el modelo de comportamiento.

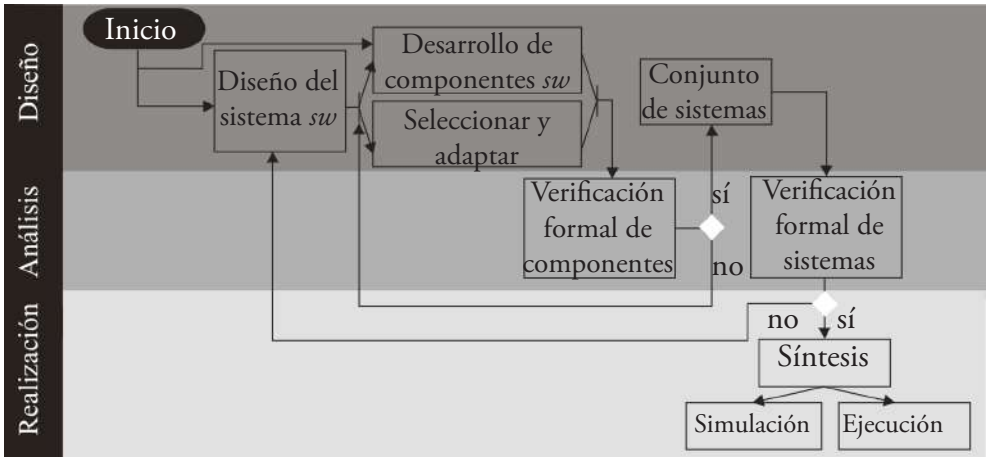


Figura 2.9. El proceso de desarrollo SaveCCT [Sentilles *et al.*, 2009]

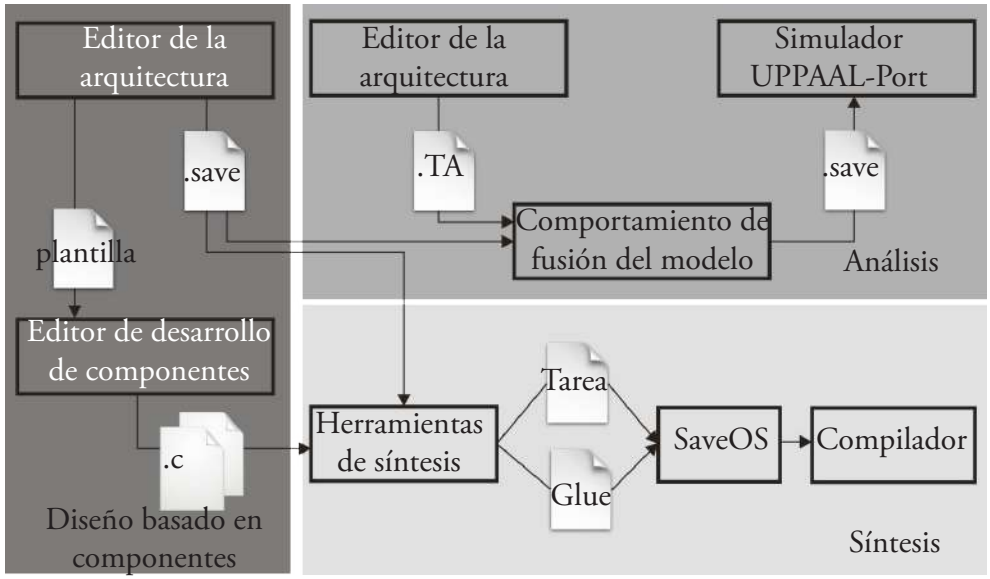


Figura 2.10. Vista general de la estructura de Save-IDE [Sentilles *et al.*, 2009]

2.5.3. *RESOURCE MODEL FOR EMBEDDED SYSTEMS (REMES)*

La importancia del conocimiento sobre los recursos en los SE está creciendo rápidamente pues la limitada disponibilidad de éstos representa la principal limitante de la introducción de nuevas características del producto y aplicaciones. Así, el análisis sistemático del consumo de recursos en un SE debe incluir los caminos de representación semántica de varios tipos de recursos, ya sean de tipo continuo (como la energía), o de naturaleza discreta (la memoria, puertos de I/O).

Una meta del análisis representativo es verificar la propiedad de viabilidad de los recursos. De acuerdo a [Seceleanu, Vulgarakis & Petterson, 2009], es posible afirmar que la composición de las necesidades de recursos para los componentes, se mantiene dentro de los disponibles proporcionados por la plataforma de ejecución, o que existe una ruta de ejecución que utiliza no más de los recursos disponibles para comportarse correctamente. En la práctica, a menudo puede ser necesario sustituir un componente por otro que tiene la misma funcionalidad. Es altamente deseable que el análisis del uso de los recursos aparezca en una fase al inicio del diseño. Esto permite la realización de un número potencialmente elevado de experimentos de diseño, sin aumentar los costos. Además, el análisis de los recursos puede orientar a los diseñadores en la toma de decisiones, como la selección de los componentes adecuados de un repositorio, o elegir entre varios modelos admisibles de diseño. REMES es un marco de modelado y análisis de aplicaciones que utiliza técnicas para realizar el análisis cuantitativo de la viabilidad y el análisis de recursos optimo-peor. REMES está diseñado específicamente para SE, pero generalmente es adecuado también para sistemas reactivos. El modelo da soporte a los recursos abstractos discretos y continuos; así, los recursos genéricos pueden ser modelados de este modo, incluyendo la memoria, puertos, energía, CPU, y buses.

2.5.3.1. OBJETIVO

El modelo REMES para SE introduce el modelado y análisis formales de los recursos empotrados tales como el almacenamiento, la energía, la comunicación y el cómputo. Este modelo es un lenguaje de comportamiento basado en una máquina de estados con soporte para el modelado jerárquico, anotaciones de recursos, tiempo continuo, y nociones de puntos explícitos de entrada y salida que resultan convenientes para el modelado de SE basado en componentes. El objetivo principal de REMES es reducir la brecha entre el modelado de arquitectura (por ejemplo, los Lenguajes para la Descripción de la Arquitectura (ADL)) y los modelos formales de análisis (por ejemplo, la programación de autómatas).

2.5.3.2. ESTRUCTURA

REMES está enfocado a describir el comportamiento racional de los recursos que interactúan entre los componentes empotrados. REMES depende en gran medida del lenguaje de modelado CHARON, utilizado para especificar la comunicación de los agentes en los SE. REMES agrega información sobre el consumo de los recursos y su tasa, así como otras construcciones que facilitan su aplicación al modelado funcional y extra-funcional de la conducta (en tiempo real) y los sistemas basados en componentes. La Figura 2.11 muestra que en REMES el comportamiento interno de un componente es descrito a través de un modo.

Se denomina 'modo atómico' si éste no contiene a su vez algún submodo, y 'compuesto' si contiene a su vez submodos. De manera similar que en CHARON, los datos son transferidos entre los modos a través de una interfaz de datos bien definida, es decir, variables de tipo global, mientras que el control (discreto) se realiza a través de una interfaz de control bien definida que consiste de puntos de entrada y salida.

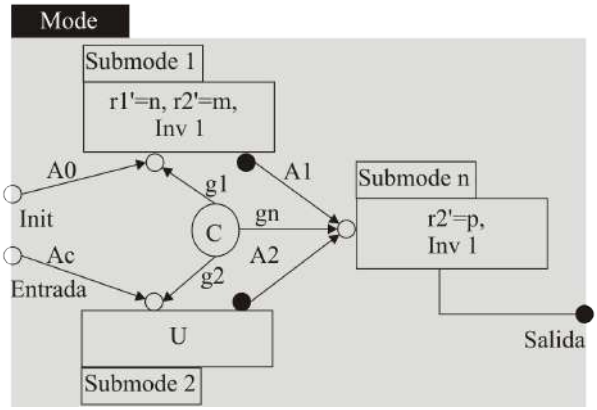


Figura 2.11. Modo compuesto de REMES [Seceleanu, Vulgarakis & Petterson, 2009]

2.5.4. *RAPID OBJECT-ORIENTED PROCESS FOR EMBEDDED SYSTEMS (ROPES)*

ROPES (*Rapid Object-Oriented Process for Embedded Systems*) se propone como un proceso completo para el desarrollo de SE. La metodología en que se enmarca ROPES, emplea directamente UML y añade un proceso de desarrollo que se define como iterativo (o en espiral) y que está basado en modelos [Powel, 2004]. ROPES está compuesto de diversas tendencias de la Ingeniería de Software, tales como el análisis de riesgos y la calidad del software.

2.5.4.1. OBJETIVO

ROPES está diseñado para emplearse en cualquier tipo de sistemas pero es especialmente útil para el desarrollo de SE pues hace énfasis en las características propias de las aplicaciones empotradas y de tiempo real. El objetivo de ROPES es producir aplicaciones software con un mínimo esfuerzo, sin fallos y con la máxima 'pronosticabilidad' en su desarrollo; por lo cual se centra en mejorar y ordenar el proceso haciendo una rápida realimentación, verificación temprana, reducción del riesgo, y mejora en la previsibilidad del proyecto en términos de esfuerzo y tiempo de programación.

2.5.4.2. ESTRUCTURA

El proceso se organiza en cuatro grandes fases: análisis, diseño, traducción y pruebas, siendo los artefactos resultantes de cada fase, modelos o diagramas UML que se refinan o corrigen en las fases siguientes. A continuación se resumen brevemente las cuatro fases de ROPES:

1. Análisis: esta fase cumple principalmente con dos propósitos: capturar y refinar los casos de uso, y asignar un conjunto de objetos que suministren el comportamiento del sistema. La fase se compone de:

- **Análisis de requisitos:** consiste en la especificación de caja negra de los requisitos funcionales y no funcionales del sistema.
- **Análisis del sistema:** determina la división de responsabilidades entre el hardware y el software, la arquitectura de alto nivel del sistema y los algoritmos de control necesarios.
- **Análisis de objetos:** se divide en dos sub-fases, en una se determinan los modelos estructurales de los objetos que han de componer el sistema y en la otra se modela su comportamiento mediante colaboraciones o diagramas de secuencia.

2. Diseño: optimiza al modelo de análisis, además de que se llevan a cabo los diseños de la arquitectura, de los mecanismos y del esquema detallado; el diseño define una única solución particular, que es óptima en algún sentido. La fase se compone de:

- **Diseño de arquitectura:** apunta a las decisiones estratégicas del diseño que afectan a la aplicación en su conjunto, tales como el modelo de despliegue, recursos de tiempo de ejecución y la concurrencia.
- **Diseño de mecanismos:** optimiza el comportamiento de las colaboraciones.
- **Diseño detallado:** optimiza el sistema de cara a su implementación.

3. Traducción: comprende la generación de código ejecutable a partir del diseño del sistema, implica tanto el código de la aplicación como el necesario para la verificación independiente de cada uno de sus objetos.

4. Pruebas: se trata de verificar la conformidad de la aplicación, ya sea para encontrar defectos o para observar un cierto nivel funcional. Incluye pruebas de integración y de validación.

Las cuatro fases antes descritas se encadenan de manera cíclica hasta lograr el nivel de aceptación y calidad deseadas. ROPES define los diagramas UML que deben desarrollarse en cada fase y que serán el resultado de esta, así como la información inicial de la siguiente. Así, de cada fase se obtiene como resultado artefactos (diagramas UML) que se refinan o corrigen en las siguientes. El proceso ROPES utiliza tres diferentes escalas de tiempo simultáneas [Powel, 1999]:

- El macrociclo (véase Figura 2.12) que hace referencia al desarrollo completo del proceso, desde que se inicia su especificación hasta que se entrega el producto. Dentro de él se desarrollan paralelamente las cuatro fases que estructuran a ROPES. Su objetivo es dirigir la evolución de los prototipos que se programan, de forma que los problemas sobre conceptos claves, requisitos, arquitectura y tecnología sean resueltos sucesivamente.

Figura 2.12. Iteración de cada microciclo en el proceso ROPES [Powel, 2004]

- El microciclo que tiene un ámbito más reducido de tiempo que el macrociclo y su objetivo es el desarrollo de un nuevo prototipo con una funcionalidad limitada que resuelva actividades consideradas de riesgo en el proceso de desarrollo.
- El nanociclo que tiene el ámbito más reducido y cuyo objetivo es modelar, ejecutar o establecer ideas que se necesitan. Su duración no debe exceder los dos días. La razón por la que se introduce esta fase es la conveniencia de probar continuamente y estar seguro de todas las ideas que se incorporan al proyecto.

Como herramienta de soporte para la elaboración y gestión de los artefactos, ROPES propone el uso de IBM® Rational ® Raphsody , en parte debido a que con esta herramienta se automatiza la generación del código.

2.5.5. *MODEL-DRIVEN DESIGN OF EMBEDDED SYSTEMS (ModES)*

Los enfoques basados en MDE[2] (*Model Driven Engineering*) se han propuesto como una metodología eficiente para el diseño de SE. Uno de los enfoques de la MDE es MDA (*Model Driven Architecture*), que es un marco propuesto por la OMG[3] para el desarrollo de software apoyado por modelos en diferentes niveles de abstracción. En un enfoque MDA, un sistema es modelado usando un Modelo Independiente de Plataforma (PIM), que se transforma en un Modelo Específico de la Plataforma (PSM), considerando un Modelo de Plataforma (PM). Los lenguajes utilizados para expresar estos modelos se definen por medio de meta-modelos que son capaces de definir la sintaxis abstracta y concreta, así como la semántica operacional de los lenguajes de modelado. MDE permite a los desarrolladores construir soluciones usando abstracciones, tales como las que proporcionan los lenguajes visuales adaptados al dominio de la solución.

En MDE, no se considera únicamente la generación de código como objetivo sino que además se incluyen operaciones tales como la transformación de artefactos software, su integración o composición, la interoperabilidad, la reutilización y la trazabilidad. El paradigma MDE tiene dos ejes principales: por un lado hace énfasis en la separación entre la especificación de la funcionalidad esencial del sistema, y por otro se enfoca en la implementación de dicha funcionalidad usando plataformas tecnológicas específicas. Existen muchos esfuerzos recientes de investigación sobre el diseño de SE basados en MDE. A través del enfoque MDE el diseñador puede especificar el sistema con un alto nivel de abstracción, buscando la mejor alternativa para desarrollarlo sin depender de costosos ciclos de evaluación de síntesis y de simulación, y generar el código fuente y scripts para automatizar y facilitar la transición de las fases de diseño inicial hasta la fase de ejecución.

2.5.5.1. OBJETIVO

ModES [Riccobene *et al*., 2006]es una metodología y un conjunto de herramientas que aplican el enfoque MDE para el diseño de SE.

2 MDE es una metodología de desarrollo de software que se centra en la creación de modelos o abstracciones. Esta metodología tiene el propósito de aumentar la productividad mediante la maximización de la compatibilidad entre sistemas, simplificando el proceso de diseño, y promoviendo la comunicación entre los individuos y equipos que trabajan en el sistema.

3 Object Management Group o Grupo de Gestión de Objetos, es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

La metodología incluye una infraestructura de meta-modelos y herramientas para apoyar las tareas de diseño basadas en modelos tales como la exploración del espacio de diseño, la estimación, la generación de código, y la síntesis de hardware.

2.5.5.2. ESTRUCTURA

ModES se basa en un metamodelo que puede ser utilizado por varias tareas durante el diseño de SE. Desde la especificación hasta la generación y la síntesis del software/hardware, las herramientas de diseño pueden tener acceso a la infraestructura del metamodelo para generar los artefactos específicos del diseño, que servirán como insumos para las tareas del diseño. Los artefactos resultantes pueden ser almacenados en el repositorio de metadatos para continuar con el flujo de diseño. La Figura 2.13 muestra que ModES está compuesto por cuatro metamodelos internos cuyo objetivo es permitir su aplicación independiente y el modelado de la plataforma. La aplicación interna del metamodelo consiste en obtener un modelo uniforme con una sintaxis bien definida y una semántica operacional. Mientras que la plataforma interna del metamodelo corresponde a una taxonomía de los elementos de hardware y software con las propiedades de la caracterización. El mapeo del metamodelo describe las reglas utilizadas para transformar los casos de "aplicación interna del metamodelo" y la "plataforma interna del metamodelo" en una instancia de la aplicación del metamodelo y define cómo las funciones de aplicación se dividen entre los componentes arquitectónicos. El enfoque MDE se utiliza para generar las representaciones internas realizando transformaciones a base de modelos y apoyándose en QVT (*Query*, Vistas, Transformaciones). En la versión actual, se utilizan las características de *Eclipse Modeling Framework* (EMF) para implementar la infraestructura de metamodelado.

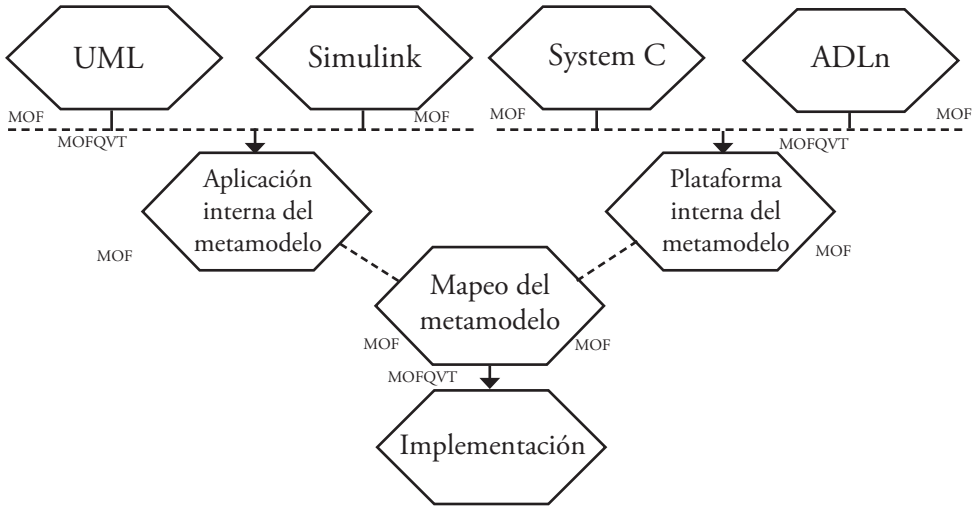


Figura 2.13. Estructura del meta-modelo [Riccobene *et al.*, 2006]

ModES utiliza un repositorio para almacenar los componentes hardware/software disponibles, la metainformación sobre ellos, así como los artefactos resultantes de aplicar el metamodelo. La metainformación puede corresponder a las propiedades físicas, tales como el rendimiento o el consumo de energía, las dependencias lógicas entre los componentes, o la información de versiones. Esta información será utilizada para apoyar las tareas de diseño, como la estimación, simulación y diseño.

2.5.6. *SIMPLIFIED PARALLEL PROCESSES (SPP)*

El desarrollo de SE, al igual que el desarrollo de software, presenta múltiples y variados conflictos, y muchas empresas ignoran una posible solución con la Mejora del Proceso Software (*Software Process Improvement*, SPI). Modelos como CMM/CMMI establecen las pautas para desarrollar software de calidad asumiendo que las empresas poseen recursos y tiempo para mejorar el proceso de software. Sin embargo puede necesitarse de una gran cantidad de recursos si cada área de proceso clave de CMM/CMMI ha sido ignorada. CMMI, describe en detalle la administración, mientras que los procesos de desarrollo tales como el desarrollo de los requisitos, el diseño, la codificación, las pruebas y el mantenimiento son sólo introducidos en un proceso llamado "Ingeniería de Producto" que sólo indica "qué" debe mejorarse sin indicar el "cómo" realizar las actividades de mejora de los procesos software, además de ser lineamientos demasiado generales como para ser adecuados para diferentes tipos de empresas. Es por esto que en el trabajo realizado por [Jun, Rui & Yi-min, 2007] se adapta y se presenta un método practico para orientar el desarrollo de SE bajo el enfoque de SPI, relacionado con el modelo CMMI.

2.5.6.1. OBJETIVO

El objetivo de *Simplified Parallel Process* (SPP) consiste en proporcionar una solución orientada a SPI y relacionada con el modelo CMMI en sus Niveles 2 y 3, y enfocada a empresas medianas. Esta investigación intenta proveer especificaciones de desarrollo técnico para los SE.

2.5.6.2. ESTRUCTURA

El modelo SPP consta de tres capas: el nivel superior consiste en los procesos para la gestión del proyecto, la capa intermedia está integrada por los procesos para el desarrollo del proyecto, y el nivel inferior sirve de apoyo a los procesos del proyecto. De acuerdo a los autores, SPP tiene las siguientes ventajas:

- La gestión de proyecto, el desarrollo y el apoyo son progresos dependientes y relacionados. Cada elemento en la organización sabe cuando hacer lo que debería hacer según la especificación.
- Se añade un inicio y terminación de la gestión de proyecto y los procesos de desarrollo son mejorados y detallados.
- SPP puede ser adaptada fácilmente a cualquier tipo de empresa.

SPP especifica los procesos para el desarrollo de SE y los incluye en forma de:

- Especificaciones técnicas de desarrollo para definir las etapas de los procesos, de las actividades y subactividades que conforman SPP.
- Directrices que indican cómo se debe realizar cada fase y las especificaciones pertinentes.
- Plantillas para ilustrar los formatos de los distintos documentos e indicar la forma de cómo realizarlas.

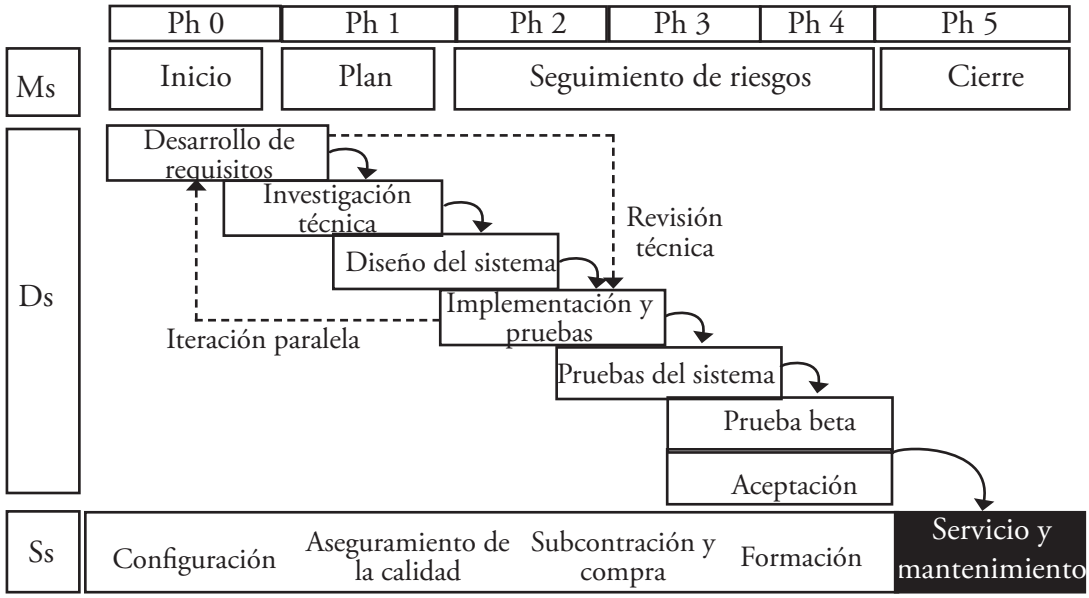


Figura 2.14. Modelo SPP [Jun, Rui & Yi-min, 2007]

La Figura 2.14 muestra la relación entre las áreas de procesos clave y el ciclo de vida del producto. Con el modelo SPP, el ciclo de vida del producto se divide en seis fases (véase Figura 2.14). Desde Ph0 a Ph5 hay 16 áreas de proceso clave, más de 40 procedimientos y 60 plantillas para documentar. SPP además aborda los procesos relacionados con el desarrollo de hardware.

2.5.7. *PRODUCT FOCUSED SOFTWARE PROCESS IMPROVEMENT*

Product Focused Software Process Improvement es el resultado de un trabajo de tesis doctoral de la *Eindhoven University of Technology, Netherlands.* La tesis se enfoca principalmente en la aplicación de SPI en el dominio del software empotrado. En el trabajo se presenta entre otras cosas un estudio sobre: los productos empotrados y su calidad, el proceso de software empotrado y la calidad de producto, y las fortalezas y debilidades de las metodologías de SPI para software empotrado. La investigación tuvo una duración de cuatro años, tiempo en el que se realizaron diferentes casos de estudio [Solingen, 2002].

2.5.7.1. OBJETIVO

El objetivo de la tesis doctoral es el desarrollo del modelo conceptual RMP centrado en la Mejora del Proceso Software para el desarrollo de productos empotrados. Así como también la descripción de un conjunto de directrices para el uso práctico de este modelo.

2.5.7.2. ESTRUCTURA

Product Focused Software Process Improvement se basa en un modelo conceptual llamado RPM. El modelo es el resultado de la combinación de tres áreas de trabajo:

- La *Ingeniería de Requisitos* es el proceso es el proceso en el que se describe lo que un producto debe hacer. El objetivo del proceso es darle a todas las partes una explicación escrita del problema. Por lo tanto la Ingeniería de Requisitos trata los principios, métodos, técnicas y herramientas que permiten obtener, documentar y mantener los requisitos. La importancia de la Ingeniería de Requisitos en *Product Focused Software Process Improvement* se debe al impacto que tienen los requisitos en la calidad del producto. La tesis hace uso de una definición práctica de Ingeniería de Requisitos que indica: "*es el proceso que recoger los deseos de todos los interesados en el producto y los transforma en una especificación de la calidad del producto, constante, inequívoca, y medible*". La Ingeniería de Requisitos proporciona una especificación de la calidad del producto y abarca los aspectos de: requisitos de calidad, percepción de la calidad que tienen los interesados o usuarios, y medición de la calidad. En la Figura 2.15 se presenta el proceso definido para Ingeniería de Requisitos.

- La *Ingeniería del Proceso* es la disciplina de ingeniería para diseñar, construir y adaptar los métodos, técnicas y herramientas para el desarrollo de un producto de software específico. Su importancia radica en el hecho de que el proceso de desarrollo más adecuado para un proyecto específico tiene que ser diseñado para su situación específica. Dentro de la tesis, Ingeniería del Proceso, "*es el diseño de un proceso medible para el desarrollo de un producto específico que obedece a la especificación de calidad de producto. Un proceso de desarrollo contiene un conjunto de acciones de proceso con efectos explícitos esperados sobre la calidad de producto*". En la Figura 2.16 se presenta el proceso definido para Ingeniería del Proceso.

- El *Programa de Medición* consiste en el diseño e implementación de un conjunto de procesos, productos y métricas de recursos, para alcanzar los objetivos predefinidos dentro de una organización. La importancia radica en el hecho de que las mediciones son objetivas y por lo tanto puede ser utilizado como un mecanismo de retroalimentación y evaluación. En *Product Focused Software Process Improvement* se define de marea práctica como "*el diseño e implementación de un conjunto de procesos, productos, recursos y métricas, para evaluar la calidad del producto y evaluar la relación producto-proceso*". En la Figura 2.17 se presenta el proceso definido para el Programa de Medición.

A partir de los casos de estudio realizados en [Solingen, 2002], se demostró que el modelo conceptual RPM es de gran beneficio para los proyectos de desarrollo industrial. El uso de este modelo en la práctica está totalmente integrado a los proyectos de desarrollo y a la mejora de los procesos, haciendo frente a los objetivos de calidad de los productos. El modelo se apoya de tres áreas de trabajo y contiene tres productos de trabajo que se relacionan directamente, tal y como se muestra en la Figura 2.18:

- **Especificación de la calidad del producto:** son los requisitos de calidad del producto debidamente documentados, especificados de un modo completo, constante, inequívoco, y medible.
- **Modelo de proceso de desarrollo:** es el modelo específico del proyecto que se integra por los pasos necesarios para realizar un producto empotrado.

- **Mediciones de productos y procesos:** son los datos y análisis realizado por el equipo de desarrollo en cuanto a la conformidad de las especificaciones de la calidad del producto, o en cuanto a la eficacia de acciones de proceso específicas.

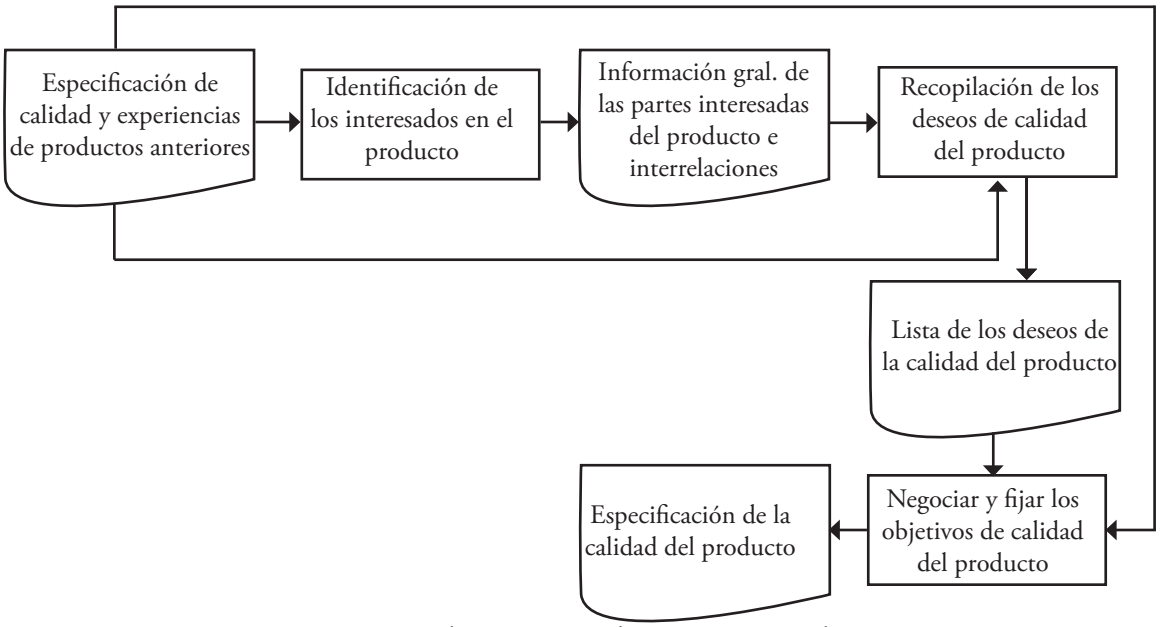


Figura 2.15. Proceso de Ingeniería de Requisitos [Solingen, 2002]

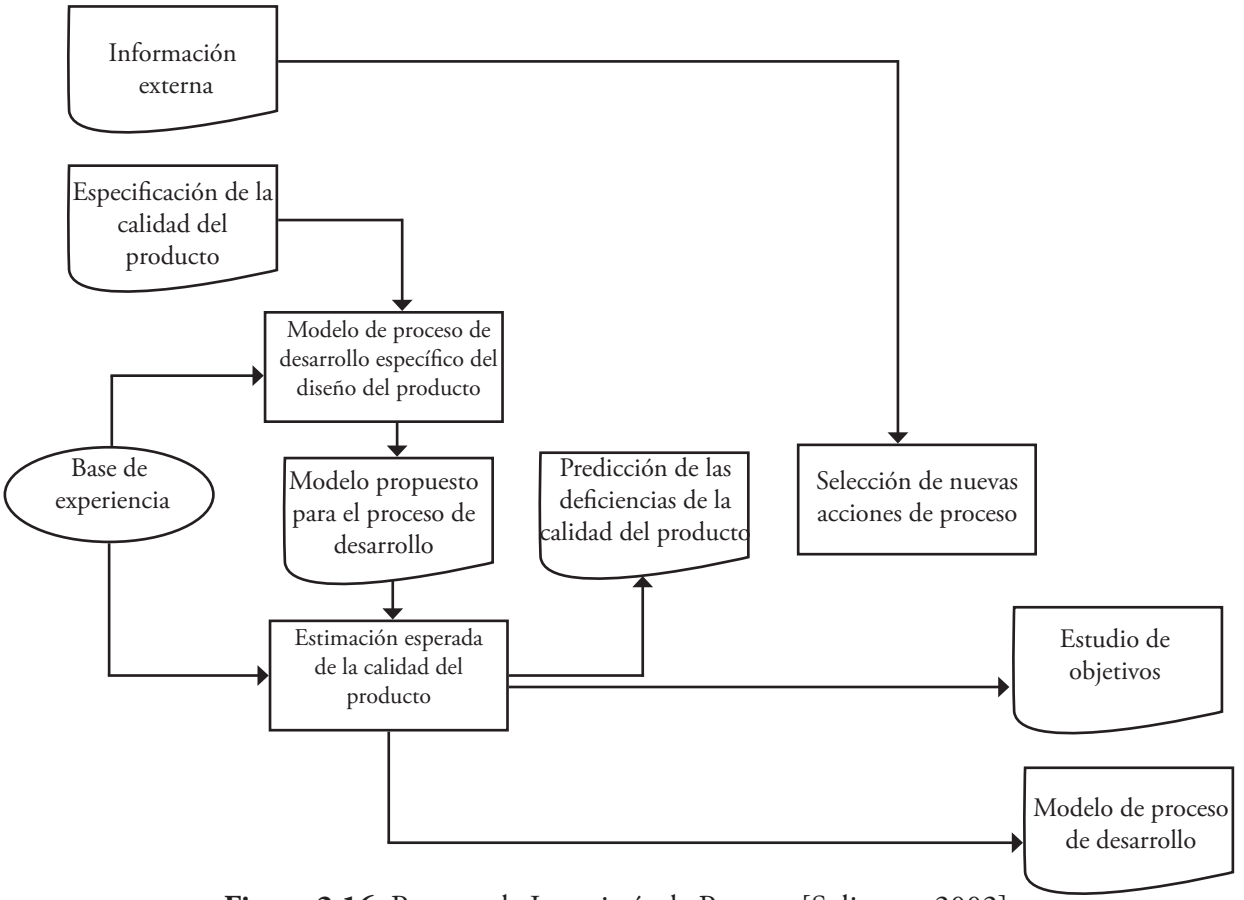


Figura 2.16. Proceso de Ingeniería de Proceso [Solingen, 2002]

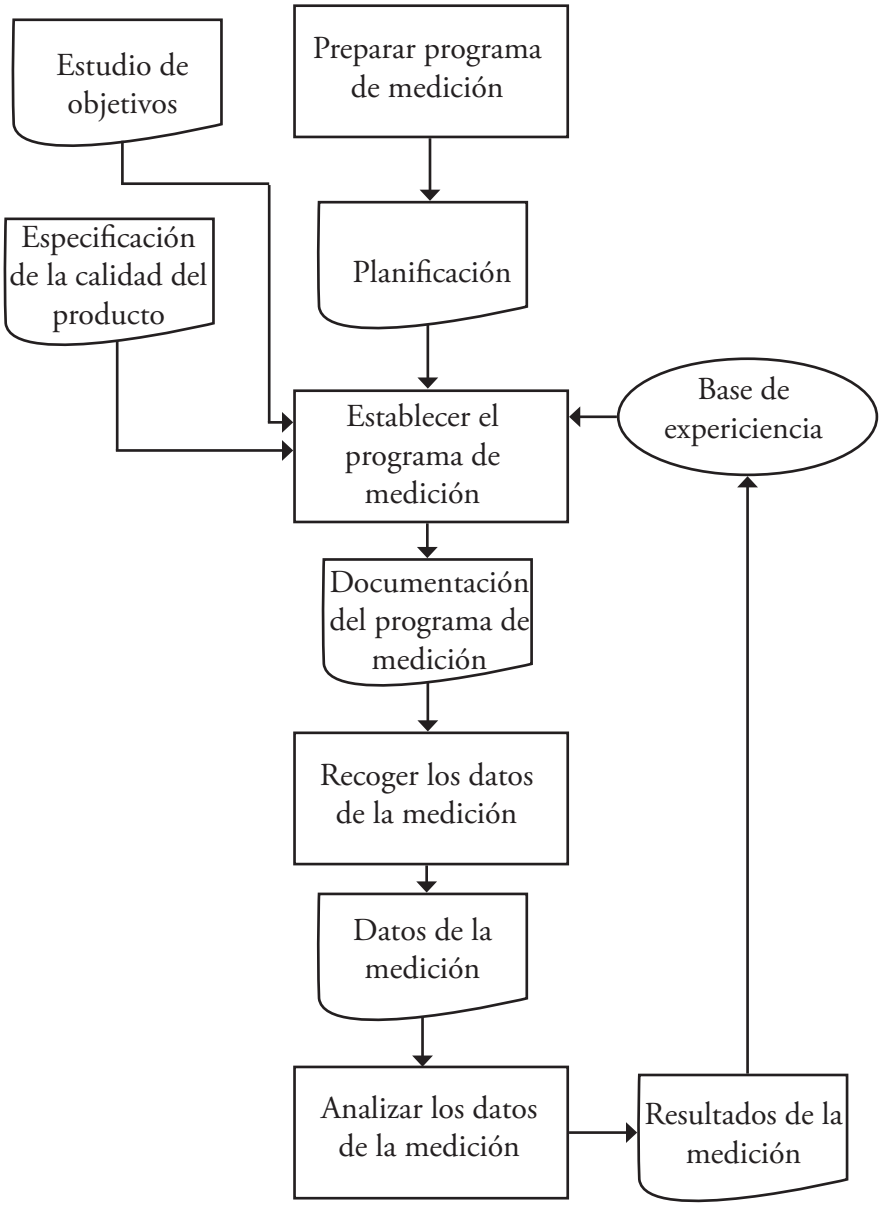


Figura 2.17. Proceso del Programa de Medición [Solingen, 2002]

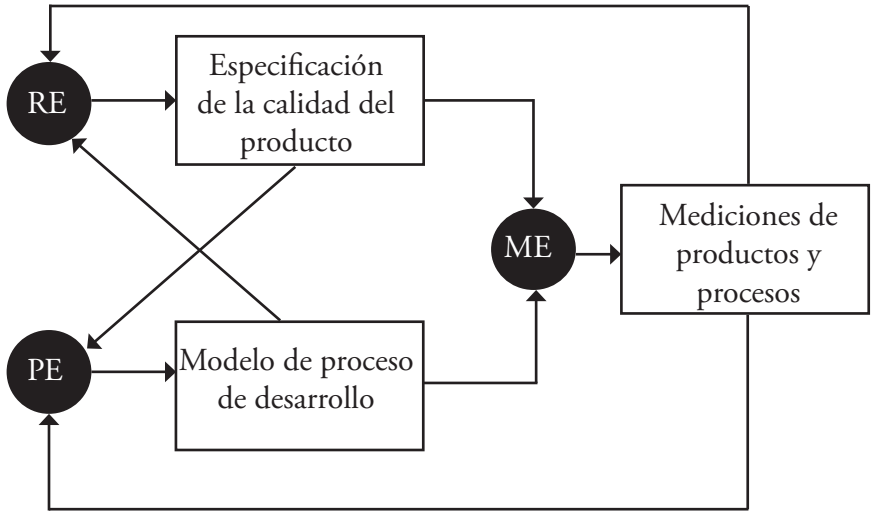


Figura 2.18. Modelo RPM [Solingen, 2002]

2.6 PRINCIPALES HALLAZGOS SOBRE LAS PROPUESTAS PARA EL DESARROLLO DE SISTEMAS EMPOTRADOS

Las metodologías analizadas son producto del esfuerzo de la industria e instituciones académicas, cuyo objetivo es incorporar formalidad al "proceso" de diseño de los SE.

Con el objetivo de identificar características comunes entre las metodologías actuales, se formuló un conjunto de criterios comparativos que a continuación son descritos de manera breve:

1. **Ámbito de aplicación:** este criterio se estableció para delimitar el uso de las metodologías, pues aunque se seleccionaron metodologías con características propias para SE, algunas de las metodologías son diseñadas y/o aplicadas a sistemas en general, sistemas empotrados, o software. Con este criterio se identificarán aquellas metodologías propias de los SE.

2. **Basado en:** de acuerdo al estudio exploratorio realizado, las metodologías actuales proporcionan soluciones orientadas principalmente al uso de Componentes (C), Herramientas (H) o a la Mejora del Proceso Software (SPI); todas con el fin de mejorar el proceso de desarrollo. Mediante este criterio se identifica la orientación hacia el objetivo de la metodología producto de esta tesis.

3. **Puntos a favor:** especifica la mejora real y/o beneficios que ofrece la metodología respecto al diseño de los SE.

4. **Puntos en contra:** especifica los inconvenientes de la metodología en el dominio de los SE.

La Tabla 2.3 resume los principales hallazgos sobre el análisis de cada herramienta.

Tabla 2.3. Tabla comparativa de las metodologías actuales para el desarrollo de SE

Metodología	Ámbito SG | SE | SW	Basado en C | H | SPI	Puntos a favor	Puntos en contra
ESCM	—	—	Reutilización, contratos a nivel interfaz, atención en los atributos no funcionales. Número de fases: 4.	Se enfoca únicamente en el desarrollo de software empotrado y no contempla el desarrollo de hardware.
SAVE-IDE	—	—	Reutilización, fases de desarrollo, herramientas asociadas. Número de fases: 4.	Falta de modelado del comportamiento de los componentes internos y externos, así como de los recursos del SE.
REMES	—	—	Modelado, gestión de recursos, lenguaje CHARON. Modelo formal.	Depende mucho de los recursos obtenidos para formular el diseño. No incorpora fases.
ROPES	—	—	Realimentación, verificación temprana, busca del mínimo esfuerzo para el desarrollo, reducción de fallos. Número de fases: 4.	No propone una estrategia que permita integrar la planificación en el proceso de desarrollo.
ModES	—	—	Abarca las áreas de especificación hasta la generación y la síntesis de software/hardware de SE.	Orientada únicamente al software empotrado y existe poca información. No incorpora fases.
SPP	—	—	Se integra por Especificaciones técnica, directrices y plantillas. Núm. de fases: 6.	Orientada únicamente al software empotrado.
Focused Software Process Improvement	—	—	Especificación de la calidad del producto. Establece directrices para las tres áreas de trabajo bajo las que está diseñada	No cubre todos los aspectos del desarrollo de un sistema empotrado. No incorpora fases.

3

MEJORA AL PROCESO SOFTWARE

En un futuro muy cercano, el éxito de los SE dependerá de la habilidad de realizar efectivamente su desarrollo y de la calidad del software que contienen. El desarrollo de software en SE es vital pues proporciona gran parte del valor del producto. Sin embargo, hablar de calidad implica contar con parámetros para establecer los niveles que el producto debe alcanzar para que sea considerado de 'calidad'.

En los estudios realizados por ITEA[4] sobre el desarrollo de software en el mundo, se muestra que la calidad y la productividad de la industria no han sido capaces de mantenerse al nivel de las necesidades de la sociedad. Por lo que resulta necesario que la producción de software se convierta en un proceso disciplinado y aceptado por todos los involucrados en el desarrollo. Por lo anterior resulta necesario definir, gestionar, optimizar e institucionalizar el proceso de desarrollo de SE mediante la Mejora del Proceso Software. Una de las principales metas de SPI es producir software de calidad, en tiempo, dentro del presupuesto y con la funcionalidad requerida [Serrano, Montes de Oca & Cedillo, 2006]; muy acorde con uno de los principales retos de los SE.

Dos nuevos conceptos aparecen en este contexto: el proceso software y la mejora del proceso software. El *Software Engineering Institute* (SEI) define un proceso software como "*un conjunto de actividades para desarrollar y mantener el software y los productos asociados (documentos de diseño, casos de prueba, manuales de usuario, etc.) y gestionar su producción*" [CMMI, 2002]. Por otra parte la mejora del proceso software "*es una iniciativa que permite valorar el estado actual de una organización en relación con sus procesos, establecer soluciones y medir si dichas soluciones mejoraron la situación*" [Baskerville, Pries-Heje & Ramesh, 2007]; lo que generará una gran cantidad de conocimiento en forma de activos intangibles de la organización [Capote *et al.*, 2009].

La Figura 3.1 muestra el complejo y cambiante entorno de desarrollo software y la relación con SPI. La naturaleza especial del proceso de software hace más difícil la implantación de la calidad, ya que no es un proceso de producción típico ni de ingeniería pura, sino que en buena medida es creativo y está basado en descubrimientos que dependen de la comunicación, coordinación y cooperación dentro de marcos de trabajos predefinidos [URL-3]. Un desafio clave en nuestra sociedad; dado que de acuerdo a Conradi [Conradi *et al.*, 2003], SPI cubre la evaluación, el perfecionamiento y la inovacion del proceso. Así, SPI es el concepto que agrupa las iniciativas de mejora de calidad y/o conjunto de actividades con las que la organización trata de alcanzar un mejor rendimiento en el costo del producto, un menor tiempo de comercialización y calidad del producto, mejorando el proceso de desarrollo del mismo. Lo que da lugar al concepto de programa de mejora de procesos software el cual tiene como propósito el mejoramiento continuo de los procesos a través de un ciclo de vida iterativo, dicho programa contribuye al aumento gradual de la satisfacción del cliente, a la obtención de un producto de calidad y a la generación de lecciones aprendidas [Capote *et al.*, 2008].

En los programas de SPI se involucran diferentes tipos de modelos[5]. Los modelos de proceso describirán la infraestructura, actividades, ciclo de vida y consideraciones prácticas para la evolución de los procesos [Pino *et al.*, 2006]. En la actualidad, existe una amplia gama de modelos y estándares de referencia para mejorar la calidad de los productos de desarrollo software, entre los que se encuentran: *Capability Maturity Model* (CMM) [Jalote, P., 2000], *Capabality Maturity Model Integration* (CMMI for Development v1.2) [CMMI, 2006], BOOSTSTRAP [Kuvaja *et al.*, 1994], ISO/IEC 12007 [ISO/IEC 12207: 2008, 2008], ISO/IEC 15504 [ISO/IEC TR 15504, 2004], *Quality Improvement Paradigm* (QIP) [Basili, 1985], PDCA Cycle (*Plan-Do-Check-Act*) [Deming, 1986], Software-CMM (SW-CMM) [Paulk, 1999], Métodos Ágiles [Greene, B., 2004], PROFES [Birk *et al.*, 1998], entre otros.

4 Information Technology for European Advacement (http://www.itea2.org/).

5 Modelos o también llamados modelos de mejora. Un modelo es una colección estructurada de elementos que describen todas las características de los procesos.

Por otra parte la forma más utilizada para conducir la mejora es la adaptación del modelo IDEAL [McFeeley, 1996]. Cabe mencionar que de acuerdo al estudio presentado en [Garzás, Fernández & Piatinni, 2009], de entre todos los modelos de mejora que existen, destacan dos grandes grupos que se han convertido en los de mayor uso en la industria del software: CMMI e ISO/IEC 15504. A continuación estos dos modelos, y otros más, serán descritos a detalle.

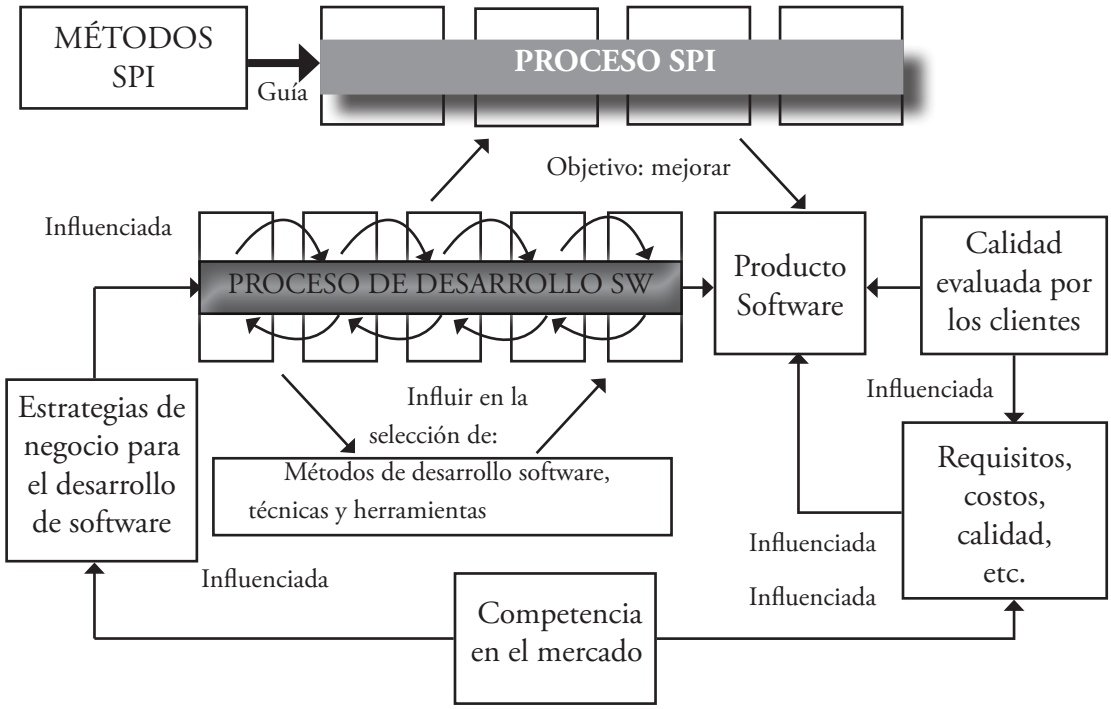


Figura 3.1. Entorno del proceso de desarrollo software y mejora del proceso [Komi-Sirviö, 2004]

3.1 *CAPABILITY MATURITY MODEL INTEGRATION (CMMI)*

De acuerdo con el SEI de la Universidad de Carnegie Mellon, el CMMI (*Capability Maturity Model Integration* v1.1, Modelo de Madurez y Capacidad Integrado v1.1) [CMMI, 2002] es un modelo para la mejora de procesos relacionados con el desarrollo de productos y de servicios que proporciona a las organizaciones los elementos esenciales para establecer procesos eficaces. CMMI representa la fusión de un conjunto de modelos orientados a la mejora de procesos de la Ingeniería de Software, Ingeniería de Sistemas, Desarrollo de Productos y Adquisición de Aplicaciones. Este modelo es el resultado de la evolución de los modelos SW-CMM (Modelo de Madurez y Capacidad para el desarrollo de Software), SECM (Modelo de Madurez y Capacidad de la Ingeniería de Sistemas) y CMM-IPD (Modelo de Madurez y Capacidad para el desarrollo Integrado de Productos).

CMMI representa un camino de mejora que permite determinar la madurez y evaluar las capacidades de la organización que desarrolla software. Así, CMMI consiste en utilizar las *mejores prácticas*, las cuales son publicadas en documentos llamados 'modelos' que cubren las actividades de desarrollo y mantenimiento del ciclo de vida del producto, desde su concepción hasta su entrega y mantenimiento.

CMMI es un modelo de integración de múltiples ingenierías que permite la definición de la madurez y capacidad de las empresas de tecnología y cuyo marco de trabajo agrupa las mejores prácticas dentro de las denominadas "constelaciones". Una constelación es una colección de componentes del CMMI que son usados para desarrollar modelos, materiales de entrenamiento, y documentos de evaluación. Es importante mencionar que CMMI describe las características de un proceso, dice qué hacer pero no define detalladamente cómo hacer el trabajo de desarrollo: su objetivo se basa en lo que se debe hacer. Sin embargo, el *Personal Software Process* (PSP) [Humphrey, 1997] y el *Team Software Process* (TSP) [Humphrey, 2000] especifican cómo se deben realizar el plan del proyecto y demás activos que permiten el éxito en la práctica [Chrissis, Konrad & Shrum, 2009]. Los modelos que se pueden generar a partir del *marco de trabajo*[6] de CMMI dependerán de los cuerpos de conocimiento que se vaya a abarcar, entre los que destacan:

- SE: *System Engineering* (Ingeniería de Sistemas)
- SW: *Software Engineering* (Ingeniería de Software)
- IPPD: *Integrated Product Process Development* (Desarrollo Integrado del Proceso y del Producto)
- SS: *Supplier Sourcing* (Relación con Proveedores)

En la actualidad existen tres áreas de interés cubiertas por los modelos de CMMI: desarrollo, adquisición y servicios [Chrissis, Konrad & Shrum, 2009]. La versión actual de CMMI es la versión 1.2; y existen tres modelos disponibles:

- CMMI para la Adquisición (CMMI-ACQ) versión 1.2, que fue liberada en noviembre del 2007. En él se tratan la gestión de la cadena de proveedores, adquisición y contratación externa en los procesos del gobierno y la industria.
- CMMI para Servicios (CMMI-SVC), que está diseñado para cubrir todas las actividades que requieren gestionar, establecer y entregar servicios.
- CMMI para el Desarrollo (CMMI-DEV) versión 1.2, que fue liberada en agosto del 2006 [CMMI, 2006]. En él se tratan los procesos de desarrollo de productos y/o servicios.

CMMI, aunque construido por una colección de modelos múltiples, está estrechamente asociado a los modelos de la Ingeniería de Sistemas y de Ingeniería de Software. Por lo tanto, ha sido adoptado primordialmente en organizaciones que desarrollan software [Chrissis, Konrad & Shrum, 2009].

3.1.1. *CAPABILITY MATURITY MODEL INTEGRATION FOR DEVELOPMENT v1.2 (CMMI-DEV v1.2)*

El modelo CMMI para Desarrollo (CMMI-DEV) es el modelo de referencia para la mejora de las diferentes áreas de proceso en los proyectos de desarrollo y de mantenimiento de software software. CMMI para Desarrollo, versión 1.2, es una continuación y actualización de CMMI versión 1.1. Los modelos de la constelación del CMMI-DEV contienen prácticas que cubren la *Gestión de Proyectos*, la *Gestión de Procesos*, la *Ingeniería de Sistemas*, la *Ingeniería de Hardware*,

6 Se refiere a la estructura básica que organiza a los componentes CMMI y los combina dentro de constelaciones y modelos CMMI.

la *Ingeniería de Software*, y otros procesos de soporte utilizados en el desarrollo y mantenimiento [CMMI, 2006]. CMMI-DEV trata las prácticas que cubren el ciclo de vida del producto desde su concepción hasta su entrega y mantenimiento aplicada a productos y servicios.

El modelo sirve de base para cualquier organización que decida seguir un camino consistente en la mejora continua, partiendo del establecimiento e institucionalización de una serie de áreas clave de proceso. CMMI-DEV debe ser adaptado al contexto y necesidades de la organización. Es uno de los modelos de mejora de la capacidad de los procesos de desarrollo software más usado en el mundo y goza actualmente de un reconocido prestigio entre las empresas que apuestan por los modelos de mejora y gestión de la calidad. El modelo CMMI-DEV es mejorado continuamente mediante las retroalimentaciones recibidas de la comunidad de interés.

3.1.1.1. PROPÓSITO DE CMMI-DEV v1.2

El propósito de CMMI es proporcionar una guía para mejorar los procesos de la organización y la habilidad para administrar el desarrollo, adquisición y mantenimiento de productos o servicios. Por lo cual el propósito de CMMI-DEV es ayudar a las organizaciones a mejorar sus procesos de desarrollo y de mantenimiento, tanto para los productos como para los servicios. Así, el modelo ayuda a las empresas productoras de software a mejorar sus procesos siguiendo un camino evolutivo que va desde un nivel inicial *ad hoc* hasta alcanzar la madurez.

3.1.1.2. ESTRUCTURA Y DISEÑO DE CMMI-DEV v1.2

CMMI-DEV v1.2 se compone de 22 áreas de proceso que se dividen en cuatro categorías para establecer niveles de madurez y de capacidad (véase Tabla 3.1). Las 22 áreas de proceso a su vez se integran por metas y prácticas. Así cada área de proceso es subdividida en un número de metas que se estructura por un conjunto de prácticas.

Tabla 3.1. Niveles de capacidad y de madurez de CMMI-DEV v1.2 [CMMI, 2006]

Nivel	Nivel de capacidad	Nivel de madurez
0	Incompleto	N/A
1	Realizado	Inicial
2	Gestionado	Gestionado
3	Definido	Definido
4	Gestionado cuantitativamente	Gestionado cuantitativamente
5	En optimización	En optimización

En la Figura 3.2 se presenta la estructura de CMMI-DEV. A continuación se explicarán brevemente sus componentes:

- *Niveles de madurez*: significa el nivel de rendimiento que se puede esperar de una organización [Kulpa & Johnson, 2008]; se aplican al logro de mejora de procesos de una organización en múltiples áreas de proceso en su representación por etapas [Chrissis, Konrad & Shrum, 2009].

Existen cinco niveles de madurez, numerados del 1 a 5 (ver Tabla 3.1), de los cuales el nivel 3 abarca todo el ciclo de vida de un proyecto de desarrollo software, incluyendo las áreas de gestión de proyectos, gestión de procesos, ingeniería y soporte. Este nivel de madurez supone que todos los procesos relacionados se han definido y estandarizado a nivel organizativo y se emplean en todos los proyectos del ámbito de aplicación.

- ***Niveles de capacidad:*** pertenecen a la representación continua y se aplican al logro de mejora de procesos de una organización en áreas de procesos individuales. Existen seis niveles de capacidad numerados de 0 a 5 (véase Tabla 3.1). En la medida en que se satisface la meta genérica y sus prácticas genéricas en cada nivel de capacidad, se obtienen los beneficios de mejora de procesos para el área de proceso correspondiente. Los niveles caracterizan a la mejora desde un estado mal definido hasta un estado que utiliza información cuantitativa para determinar y gestionar las mejoras que se necesitan para satisfacer los objetivos del negocio de una organización.
- ***Áreas de proceso:*** un área de proceso se define como un grupo de prácticas relacionadas en un área que, cuando se implementan de forma conjunta, satisfacen un grupo de objetivos considerados importantes para la mejora en esa área. Cada área de proceso está descrita por: declaración de objetivos, notas introductorias, metas (especificas y genéricas), prácticas (especificas y genéricas), subpracticas, apuntes, productos de trabajo, relaciones con áreas de proceso, etc. (véase Tabla 3.2).
- ***Metas:*** cada área de proceso tiene varias metas que debe cumplir para satisfacer los objetivos del área de proceso. Existen dos tipos de metas: específicas (metas que se refieren únicamente al área de proceso en estudio) y genéricas (metas que son comunes a múltiples áreas de proceso en todo el modelo).
- ***Prácticas:*** son actividades que deben realizarse para satisfacer las metas de cada área de proceso. Cada práctica se refiere a una meta. Existen dos tipos de prácticas: específicas (prácticas que se refieren únicamente a una meta) y genéricas (prácticas que se aplican a múltiples áreas de proceso).

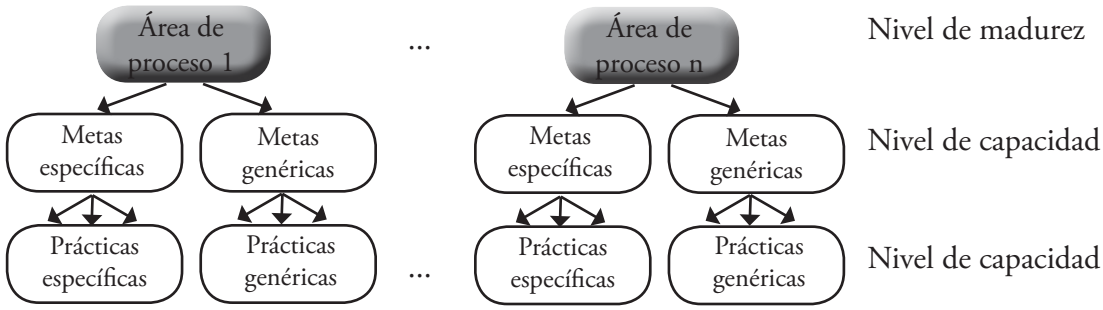


Figura 3.2. Representación de los componentes del modelo CMMI-DEV v1.2 (continua y por etapas) [CMMI, 2006]

El CMMI permite aproximarse a la mejora de procesos y a las evaluaciones usando dos representaciones diferentes: continua y por etapas.

- La ***representación continua*** es usada cuando se han identificado los procesos que necesitan ser mejorados y se conocen las dependencias existentes entre las áreas de proceso; ofrece la máxima flexibilidad y se utilizan niveles de capacidad.

Define cuatro categorías de proceso: ingeniería, gestión de proyectos, gestión de procesos, y soporte.

- La ***representación por etapas*** ofrece una manera sistemática y estructurada de aproximarse a la mejora de procesos y se utilizan cuatro niveles de madurez. Así, las áreas de proceso están organizadas por nivel de madurez.

Tabla 3.2. Áreas de proceso por nivel de madurez y nivel de capacidad [CMMI, 2006]

Área de proceso	Abreviatura	Categoria de proceso
Gestión de Requisitos	REQM	Ingeniería
Planificación de Proyectos	PP	Gestión de proyectos
Supervisión y Control del Proyecto	PMC	Gestión de proyectos
Acuerdo con Proveedores	SAM	Gestión de proyectos
Medición y Análisis	MA	Soporte
Aseguramiento de la Calidad del Producto y del Proceso	PPQA	Soporte
Administración de la Configuración	CM	Soporte
Desarrollo de Requisitos	RD	Ingeniería
Solución Técnica	TS	Ingeniería
Integración del Producto	PI	Ingeniería
Verificación	VER	Ingeniería
Validación	VAL	Ingeniería
Mejora de Procesos Organizacionales	OPF	Gestión de procesos
Definición de Procesos Organizacionales	OPD	Gestión de procesos
Formación Organizacional	OT	Gestión de procesos
Gestión Integrada del Proyecto	IPM	Gestión de proyectos
Gestión de Riesgos	RSKM	Gestión de proyectos
Análisis y Soluciones en la Toma de Decisiones	DAR	Soporte
Desarrollo del Proceso Organizacional	OPP	Gestión de procesos
Gestión Cuantitativa del Proyecto	QPM	Gestión de proyectos
Innovación Organizacional e Implantación	OID	Gestión de procesos
Análisis de Causas y Resolución	CAR	Soporte

3.1.1.3. VENTAJAS DE CMMI-DEV V1.2

Entre las ventajas de implementar y seguir el modelo CMMI-DEV, se enlistan las siguientes:

- CMMI-DEV v1.2 es un modelo de gran eficacia y está estructurado para la industria de TI [Persse, 2006], lo que lo hace apto para el estudio, diseño, desarrollo, implementación y soporte de software de aplicación y hardware.
- CMMI-DEV v1.2 propone una solución integrada y completa para las actividades de desarrollo y de mantenimiento aplicadas a los productos y a los servicios. Elimina barreras de disciplinas separadas a través de modelos integrados [Chrissis, Konrad & Shrum, 2009] que facilitan el encontrar y resolver cuestiones críticas.
- Mediante las áreas de proceso se enfatiza en prácticas importantes, tales como: Gestión de Requisitos, Ingeniería de Procesos, Análisis de las Decisiones.
- El modelo CMMI-DEV guía paso a paso la mejora a través de niveles de madurez y capacidad [Mutafelija, 2003].
- Facilita la transición del 'aprendizaje individual' al 'aprendizaje de la organización' por mejora continua, lecciones aprendidas y uso de bibliotecas y bases de datos de proyectos mejorados [Mutafelija, 2003].
- CMMI-DEV v1.2 mejora la visibilidad sobre los proyectos al recomendar las actividades a realizar; por lo tanto cada integrante del equipo sabe en qué trabajar y conoce la dirección del proyecto. A partir de esto, cada participante conoce sus responsabilidades y compromisos en el proyecto, lo que genera una mejor comunicación. Además amplía el alcance y visibilidad en el ciclo de vida del producto y las actividades de ingeniería para garantizar que el producto o servicio cumple con las expectativas del cliente.
- Las mejores prácticas del modelo CMMI facilitan que las organizaciones puedan vincular de forma más explícita las actividades de gestión e ingeniería considerando los objetivos del negocio, lo que provoca que los objetivos de procesos estén alineados con los objetivos empresariales.
- CMMI-DEV v1.2 ayudar a reducir costos debido a una mayor facilidad de las planificaciones, reducción de procesos, y acuerdos claros sobre el servicio y la funcionalidad del producto a entregar, aumento en la confiabilidad debido a la reducción considerable de errores (reduciendo el número de defectos y detección en fases tempranas del ciclo de vida), y cumplimiento de fechas.
- La constelación CCMMI-DEV v1.2 consiste de dos modelos: el CMMI-DEV v1.2 y el CMMI +IPPD [Chrissis, Konrad & Shrum, 2009].
- En el 2006, un informe del SEI, *Performance Results of CMMI-Based Process Improvement* [Gibson, Kost & Goldenson, 2006], señaló que, las organizaciones que implementaron procesos CMMI mejoraron la calidad en un 48% al tiempo en que reducían sus costos en un 34% y acordaban los plazos en un 50%.
- Principalmente, existe evidencia sobre la aplicación de los modelos CMMI al dominio de los SE.

3.1.1.4. DESVENTAJAS DE CMMI-DEV V1.2

- CMMI-DEV v1.2 exige un alto esfuerzo para su implantación, puede llegar a ser excesivamente detallado; por lo que se requiere de una mayor inversión para ser completamente implementado.

- A pesar de que el modelo especifica qué actividades se deben realizar, no da pautas claras sobre el cómo, tanto para la ingeniería como en relación a la gestión de proyectos [Sumano, 2009].
- No es un estándar internacional, sino que es un estándar de facto de uso internacional.

3.2 ISO/IEC 15504:2004

Diversos estudios continúan mostrando que la industria del software es poco madura en relación a la gestión de la calidad del software, a pesar de que la calidad es fundamental en la competitividad de las organizaciones. El proyecto SPICE promovido por la ISO, surgió de la necesidad de una norma internacional dados los múltiples modelos existentes para la evaluación y mejora del proceso software.

El proyecto se definió como un esfuerzo de colaboración internacional que debía materializarse en un nuevo estándar para la evaluación del proceso software. Se crea con el objetivo de hacer frente a la alta competencia del mercado de desarrollo de software, a la difícil tarea de identificar los riesgos, cumplir con el calendario, controlar los costos y mejorar la eficiencia y calidad. Así, ISO/IEC 15504:2004 [ISO/IEC 15504, 2004] es un estándar internacional aplicable a cualquier organización que quiere conocer y mejorar la capacidad de sus procesos, independientemente del tipo de organización, del modelo de ciclo de vida adoptado, de la metodología de desarrollo y de la tecnología utilizada.

La norma ISO/IEC 15504:2004 es una norma abierta e internacional para evaluar y mejorar la capacidad y madurez de los procesos. El rol de la norma es proveer un marco de referencia que determine las fortalezas y debilidades de los procesos, así como también mejorar los procesos de software y medir sus mejoras. Además de ser un marco de referencia para aquellos que pretenden adquirir un sistema, puesto que permite evaluar la capacidad de los proveedores de sistemas y determinar los riesgos de negocio para una empresa que considera desarrollar un nuevo producto o servicio de software.

En [Jonassen, 2003] se expresa que la norma ISO/IEC 15504:2004 no establece explícitamente los objetivos de SPICE pero si las consecuencias de éxito al aplicarla:

- Desarrollo de una estrategia para una gestión de la configuración.
- Todos los elementos generados por el proceso o proyecto serán debidamente identificados y definidos.
- Control en las modificaciones y liberación de los productos.
- El estado de los productos, así como las modificaciones de estos, serán registrados y comunicados. Así como también se asegura la integridad y coherencia de los productos.
- Control en el almacenamiento, manipulación y entrega de los productos.

3.2.1. PROPÓSITO DE LA NORMA ISO/IEC 15504:2004

ISO/IEC 15504:2004 es un marco para la evaluación, orientado a la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software que involucra la planeación, gestión, supervisión, control y mejora de la adquisición, suministro, desarrollo, operación, evolución y soporte del proceso software. La norma consiste de niveles de capacidad y ofrece las pautas para verificar si los procesos son efectivos en la consecución de sus objetivos.

3.2.2. ESTRUCTURA Y DISEÑO DE LA NORMA ISO/IEC 15504:2004

La norma internacional ISO/IEC 15504:2004 fue desarrollada inicialmente por el proyecto denominado SPICE que se orientó a elaborar normas para la evaluación de los procesos software. Sin embargo después de la primera publicación, la norma ISO/IEC 15504:2004 se convirtió en una norma para evaluar procesos en general. La norma está orientada a los procesos y se integra por las siguientes cinco partes [Tuya, Dolado & Ramos, 2007]:

- ISO/IEC 15504-1:2004. Parte 1: *Conceptos y vocabularios*. Representa una introducción general a la norma, proporcionando una guía de la misma. En esta parte se incluye el conjunto de términos definidos específicamente para la norma.
- ISO/IEC 15504-2:2003/Cor 1:2004. Parte 2: *Realizando una evaluación*. Define los requisitos que deben cumplir una evaluación para que produzca resultados repetibles, fiables y consistentes.
- ISO/IEC 15504-3:2004. Parte 3: *Guía para la realización de evaluaciones*. Establece una guía para la realización de evaluaciones de procesos, interpretando los requisitos de las partes normativas para diferentes contextos de evaluación.
- ISO/IEC 15504-4:2004. Parte 4: *Guía para el uso de la mejora y determinación de la capacidad de procesos*. Proporciona una guía para poder utilizar los resultados de una evaluación en la mejora de los procesos evaluados. La guía incluye ejemplos de la aplicación de mejoras en una gran variedad de situaciones.
- ISO/IEC 15504-5:2004. Parte 5: *Un ejemplar de modelo de evaluación de proceso*. Proporciona un modelo totalmente compatible con la parte de la norma que incluye un conjunto de indicadores que facilitan el cálculo de la capacidad de los procesos.

La Figura 3.3 presenta la arquitectura de la norma ISO/IEC 15504:2004, la cual está basada en dos dimensiones: de proceso y de capacidad. Cada implementación de la norma debe definir las dos dimensiones con los componentes que se presentan.

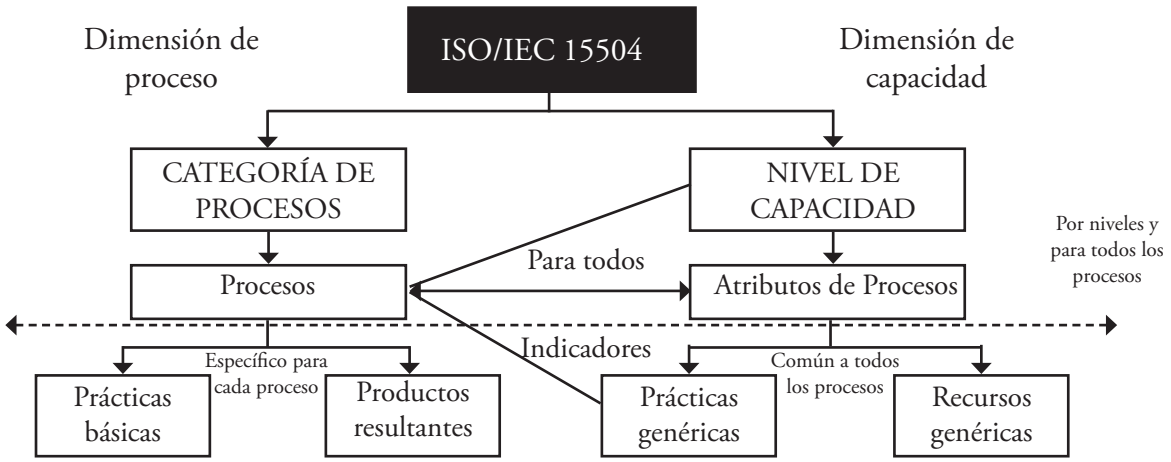


Figura 3.3. Estructura de la norma ISO/IEC 15504 [Tuya, Dolado & Ramos, 2007]

1. Dimensión del proceso: La dimensión del proceso está representada por un modelo de procesos de referencia externo y define un conjunto de estándares de proceso. Esta dimensión está determinada por los propósitos del proceso, que son los objetivos medibles del proceso. Se establece el camino para determinar la práctica actual de una organización en términos de la capacidad de los procesos y se obtienen los puntos fuertes y débiles inherentes al proceso. En esta dimensión el modelo de referencia organiza todos los procesos relevantes para la producción del software en tres grupos. Esta estructura es consistente con la norma ISO/IEC 12207 [ISO/IEC 12207, 2008]. Los grupos de procesos tienen un total de cinco categorías, tal y como se muestra en la Tabla 3.3.

Tabla 3.3. Dimensión del proceso ISO/IEC 15504 [Tuya, Dolado & Ramos, 2007]

Grupo	Categoría
Procesos del ciclo de vida primario	Cliente-Proveedor
	Ingeniería
Procesos del ciclo de vida de soporte	Soporte
	Gestión
Procesos del ciclo de vida organizacional	Organización

En la Figura 3.3 se observan que los componentes que integran la dimensión del proceso son:

- *Categoría de procesos*: los procesos se agrupan en diferentes categorías y contienen indicadores de realización, prácticas básicas y productos resultantes o entregables.
- *Procesos*: un proceso es un conjunto de actividades mutuamente relacionadas, o que interactúan, y que se establecen para conseguir un resultado.
- *Prácticas básicas*: son las actividades esenciales de un proceso específico, agrupado por categorías de procedimientos y procesos de acuerdo al tipo de actividad que direccionan.

2. Dimensión de la capacidad: Se analiza la capacidad de los procesos seleccionados con respecto a un perfil de madurez. Está caracterizada por una serie de atributos del proceso que representa características medibles necesarias para gestionar un proceso y mejorar su capacidad de funcionar. Esta dimensión se integra principalmente por los siguientes componentes:

- *Nivel de capacidad*. El modelo define una escala de valoración para la capacidad de seis niveles (ordenados del 0 al 5), los cuales son descritos en la Tabla 3.4. Los niveles de capacidad son vitales pues a partir de estos se determinan las capacidades de la organización. Estos niveles se sustentan en un conjunto de atributos del proceso que determinan el nivel. El nivel de capacidad que tiene un proceso depende de los atributos que posee y del grado con el cual se alcanzan dichos atributos.
- *Atributos de proceso*. Cada atributo valora un aspecto particular de la capacidad del proceso. Dependiendo de los valores de los atributos que alcance un proceso, éste se encontrará en una u otra posición de la escala. Un atributo de proceso representa una característica medible de cualquier proceso. Cada atributo de proceso contiene indicadores de capacidad: prácticas genéricas y recursos genéricos (herramientas, recursos, metodología, infraestructura, etc.).

- *Prácticas genéricas*. Las prácticas genéricas son aplicables a cualquier proceso que representa las actividades necesarias para administrar el proceso y mejorar su potencialidad.

Tabla 3.4. Niveles de capacidad de ISO/IEC 15504:2004 [Tuya, Dolado & Ramos, 2007]

Escala de valoración	**Atributos del proceso**	**Descripción**
Nivel 0 Incompleto	No hay atributos en este nivel.	El proceso no existe o no se consigue su propósito.
Nivel 1 Realizado	Realización del producto	Se alcanza el propósito del proceso en términos generales. Se reconoce un proceso de realización, pero no se hace de una forma planificada ni se realiza algún seguimiento.
Nivel 2 Gestionado	Gestión de realización Gestión de productos	Se obtienen los productos del proceso, pero esta vez, de acuerdo con una planificación y realizándose un seguimiento. Estos productos se ajustan a unos estándares y a unas especificaciones de requisitos prefijadas.
Nivel 3 Establecido	Definición de procesos Recursos de procesos	El proceso se realiza y se gestiona utilizando procedimientos definidos según los principios de Ingeniería de Software. Cada implementación de un proceso se hace utilizando procedimientos creados según un estándar y debidamente documentados, además, se dispone de los recursos necesarios para alcanzar los propósitos establecidos.
Nivel 4 Predecible	Medición de procesos Control de procesos	La realización del proceso se gestiona de forma cuantitativa, es decir, se recogen medidas detalladas del nivel de realización del proceso y se analizan. El proceso opera dentro de los límites definidos para alcanzar sus resultados.
Nivel 5 En optimización	Cambio de procesos Mejora continua	La realización del proceso se optimiza en forma continua, de cara a su contribución para alcanzar los objetivos de negocio de la organización.

3.2.3. VENTAJAS DE LA NORMA ISO/IEC 15504:2004

- ISO/IEC 15504:2004 es una norma internacional y es el primer modelo bidimensional, al separar los procesos y capacidad en dimensiones diferentes. Una de las dimensiones determina los procesos a ser valorados (definiendo el proceso de vida del software) y la otra representa una escala para evaluar la capacidad, por tanto son dimensiones bien determinadas y complementarias que soportan la evaluación y mejora de la capacidad y madurez de los procesos.
- Es resultado de una evolución, inició como un modelo de referencia de buenas prácticas de software, para convertirse en un marco de trabajo para evaluar múltiples modelos.

- El dominio de la norma involucra el suministro, el desarrollo, la operación y el mantenimiento de los productos de software, y establece un marco de referencia común para los procesos del ciclo de vida del software.
- Contiene procesos, actividades y tareas para aplicar durante el suministro, el desarrollo, la operación y el mantenimiento de productos de software. Los resultados serán alcanzados en una organización siguiendo prácticas detalladas para generar productos de trabajo.
- La escala elegida tiene niveles que son caracterizados por un conjunto de atributos de procesos, que a su vez, son evaluados según el grado de cumplimiento de los mismos (indicados en tanto por ciento).
- En lo referente a la calidad del software la norma considera la evaluación de procesos, la mejora de procesos y la determinación de la capacidad.
- Está alineada con el ISO/IEC 12207 e intenta proporcionar un marco en el que los enfoques existentes puedan armonizarse y funcionar como uno solo. ISOIEC 15504:2004 no está diseñado para ser utilizado por sí solo.

3.2.4. DESVENTAJAS DE LA NORMA ISO/IEC 15504:2004

- La norma ISO/IEC 15504:2004 es una norma constituida por procesos, cuya sensación a la hora de implementarse es la dificultad que presenta, pues la consideran una norma no práctica y difícil de aplicar.
- La norma carece de un mecanismo de evaluación pues solo presenta los requisitos necesarios para desarrollarlo.
- No es aplicable a los SE.

3.3 SIX SIGMA

Six Sigma es una filosofía de calidad basada en la asignación de metas alcanzables a corto plazo enfocadas a objetivos a largo plazo. Utiliza las metas y los objetivos del cliente para manejar la mejora continua a todos los niveles en cualquier empresa. Así, Six Sigma representa una métrica, una filosofía de trabajo y una meta [Feng, 2008]. Six Sigma es una métrica, pues representa una manera de medir el desempeño de un proceso en cuanto a su nivel de productos o servicios fuera de una especificación.

En base a los datos, Six Sigma lleva la calidad hasta niveles próximos a la perfección y se diferencia de otros enfoques ya que también corrige los problemas antes de que se presenten. Six Sigma mide el grado en el cual cualquier proceso del negocio se desvía de su meta, por lo que es una filosofía de trabajo que se centra en la mejora continua de procesos y productos de apoyo en la aplicación de una metodología, la cual incluye principalmente el uso de herramientas estadísticas, además de otras de apoyo. En Six Sigma se entienden las especificaciones del cliente para un producto y la confiabilidad que el cliente espera; se comprende el proceso de producción que conlleva el producto y se reduce la variación en los procesos para aumentar la confiabilidad [Lowenthal, 2002].

En base a su filosofía, el objetivo a largo plazo es el de diseñar e implementar procesos más robustos en los que los defectos se midan a niveles de solamente unos pocos por millón de oportunidades. Six Sigma como meta, es un proceso con nivel de calidad Six Sigma, lo que significa estadísticamente tener un nivel de clase mundial al no producir servicios o productos

defectuosos. Six Sigma considera primero al cliente y usa hechos y datos para impulsar la obtención de mejores resultados.

3.3.1. PROPÓSITO DE SIX SIGMA

Six Sigma es una transformación cultural, basada en el liderazgo comprometido de la alta dirección; es la puesta en práctica de una estrategia basada en mediciones que se centra en la mejora del proceso. Tiene el objetivo de desarrollar procesos, productos y/o servicios eficientes y minimizar los defectos asociados hasta un valor objetivo de excelencia para lograr la satisfacción del cliente con productos novedosos a un precio competitivo. El enfoque busca la reducción de los costos a través de la eliminación de defectos y la mejora de los procesos. Utiliza los principios de enfoque en el cliente, proceso de orientación, y liderazgo basado en métricas.

3.3.2. ESTRUCTURA Y DISEÑO DE SIX SIGMA

Para implementar la metodología de Six Sigma se debe contar con una estructura humana sólida que clasifica a los colaboradores en: campeones, maestros cinta negra, cintas negras y cintas verdes. Six Sigma proporciona medidas que se aplican tanto a las actividades de producción como a las de servicios; por lo que se cuenta con una serie de fases y pasos establecidos en una metodología. La metodología de Six Sigma se basa en lo que se conoce como DMAIC (*Define, Measure, Analyse, Improve and Control*). Así la metodología establece los siguientes pasos a realizar (véase Figura 3.4):

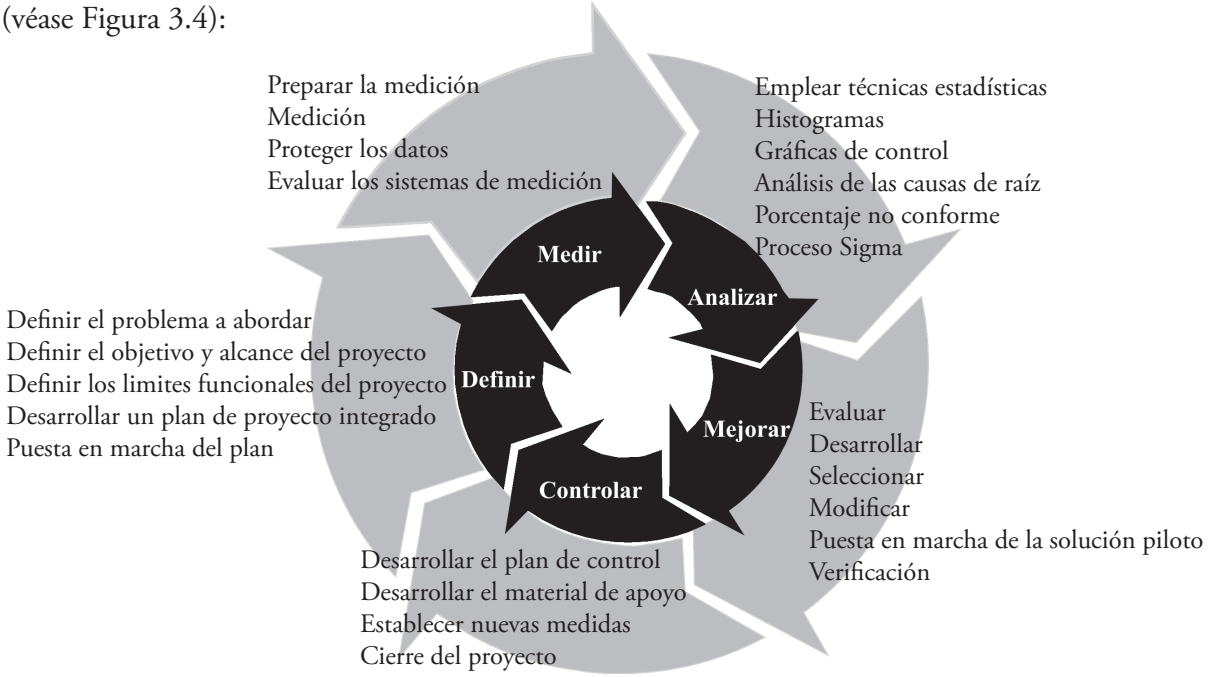


Figura 3.4. Metodología Six Sigma [Chrissis, Konrad & Shrum, 2009]

Los esfuerzos de Six Sigma se dirigen a tres áreas principales: mejorar la satisfacción del cliente, reducir el tiempo del ciclo y reducir los defectos. Las actividades en Six Sigma tratan de crear y perfeccionar los procesos que reflejan la voz del cliente, los productos y los atributos de servicio que tienen que ver con lo que el cliente quiere y necesita.

3.3.3. VENTAJAS DE SIX SIGMA

Entre las ventajas de implementar y seguir la metodología propuesta por Six Sigma, se enlistan las siguientes:

- Six Sigma es una metodología rigurosa que permite estructurar y disciplinar el mejoramiento continuo en la empresa a través del análisis estadístico para medir y mejorar el rendimiento operativo. Crea una cultura de calidad en la organización y ofrece resultados cuantificables y objetivos [Linderman, 2003].
- Entiende claramente a la empresa o negocio como un sistema interrelacionado de procesos y clientes [Miranda, 2006]. Six Sigma se enfoca en que la organización satisfaga al cliente.
- Six Sigma puede aplicarse para mejorar un proceso existente o para diseñar un proceso nuevo. Todo el mundo tiene un rol que realizar, desde los ejecutivos hasta los empleados de las líneas de producción.
- Aumenta el rendimiento y disminuye la variación de los procesos, lo que conduce a la reducción de defectos y a mejorar el proceso, la calidad del producto e incrementa el valor de la compañía o servicio.
- Six Sigma incluye una filosofía, medidas de funcionamiento, marcos de mejora y un conjunto de herramientas, todo concebido para complementar y para perfeccionar los procesos de ingeniería, de servicios y de fabricación existentes [Chrissis, Konrad & Shrum, 2009].

3.3.4. DESVENTAJAS DE SIX SIGMA

- Inversión inicial en la formación del personal, sobre todo en conceptos y herramientas estadísticas.
- Six Sigma no garantiza la calidad en sí, proporciona expectativas sobre el funcionamiento de programas que pueden relacionarse con la satisfacción del cliente, mejorando implícitamente la calidad.
- No existe evidencia de su aplicación al dominio de los SE.

3.4 BOOTSTRAP

La metodología BOOTSTRAP [Kuvaja *et al.*, 1994] es el resultado de un proyecto Europeo basado en los modelos CMM [Paulk *et al.*, 1993] e ISO 9000 [Guler, Guillén & Macpherson, 2002]. Este proyecto, desarrollado y mantenido por la organización *European Strategic Programme for Research in Information Technology* (ESPRIT), proporciona una alternativa para las organizaciones que están interesadas en mejorar su proceso de desarrollo de software y alcanzar la certificación ISO, ya que combina y realza las formas establecidas por CMM y la certificación ISO 9000 [Komi-Sirviö, 2004].

La metodología BOOTSTRAP engloba tanto la evaluación para establecer el diagnóstico de un proceso para desarrollo de software (el cual incluye a la planeación, los métodos y la capacidad de ingeniería, las herramientas y la tecnología), así como la creación de un plan de acción que defina los pasos, los detalles de la implantación y los marcos temporales para que la organización aumente su capacidad de entrega de productos y servicios de calidad. De acuerdo con [Scalone, 2006], los objetivos de la metodología BOOTSTRAP son:

- Proporcionar soporte para la evaluación de la capacidad de los procesos utilizando un conjunto de prácticas de Ingeniería del Software.
- Incluir estándares de Ingeniería del Software reconocidos internacionalmente como fuentes para la identificación de las prácticas a considerar.
- Dar soporte a la evaluación, indicando cómo el estándar de referencia ha sido aplicado en la organización evaluada.
- Asegurar la fiabilidad y capacidad de repetición de la evaluación.
- Identificar las fortalezas y debilidades de los procesos de la organización evaluada.
- Dar soporte a la creación y aplicación de un plan de mejora que genere resultados aceptables y fiables, de forma que las acciones del plan de mejora permitan alcanzar los objetivos de la organización.
- Ayudar a incrementar la eficacia de los procesos poniendo en práctica los requisitos del estándar en la organización.

BOOTSTRAP al ser una metodología de evaluación y mejora de procesos de software se basa en evaluar el nivel de capacidad y de productividad de una Unidad de Desarrollo Software (*Software Producing Unit*, SPU). De acuerdo con [Dolado, 2007], el modelo se enfoca a:

- Evaluar una SPU y sus proyectos, proporcionando perfiles analíticos para cada uno de ellos, de forma que se establezca la madurez de su proceso, identificando sus puntos fuertes y débiles.
- Deducir las áreas de mejora a partir de los perfiles analíticos, realizando un plan de alto nivel de las acciones recomendadas para conseguir la misma.
- Transformar el plan en una serie de mini-proyectos para implementar las mejoras recomendadas anteriormente.

El modelo de proceso de software utilizado por BOOTSTRAP es un modelo cíclico, compatible con el que se utiliza en ISO/IEC 15504:2004, que es equivalente al modelo IDEAL [McFeeley, 1996]. Comprende las siguientes actividades:

1. Examinar las necesidades de la empresa, realizando los primeros contactos con los responsables y delimitando la acción.
2. Iniciar el proceso de mejora, definiendo el objetivo de mejora y comenzando a desarrollar un plan.
3. Preparar y dirigir la evaluación para conocer el estado de todos los procesos de la empresa.
4. Analizar los resultados de la evaluación y establecer el plan de acción, identificando y priorizando las áreas de mejora y estableciendo el plan de acción para mejorar estos procesos seleccionados.
5. Implantar las mejoras a través de un plan de acción detallado aplicado a proyectos reales.
6. Finalizar las mejoras garantizando que se han alcanzado todos los objetivos establecidos.

BOOTSTRAP hace uso de cinco niveles de madurez para indicar el nivel en el que se encuentra la organización, por lo que emplea diferentes escalas para medir las fortalezas y debilidades y marcar las pautas de mejora. Las áreas de proceso se agrupan en tres categorías de proceso: Organización, Metodología y Tecnología. Cada categoría comprende un conjunto de áreas de proceso orientadas a obtener el mismo objetivo general.

En último lugar, cada proceso se divide en actividades y éstas en prácticas base (véase Figura 3.5) [Dolado, 2007].

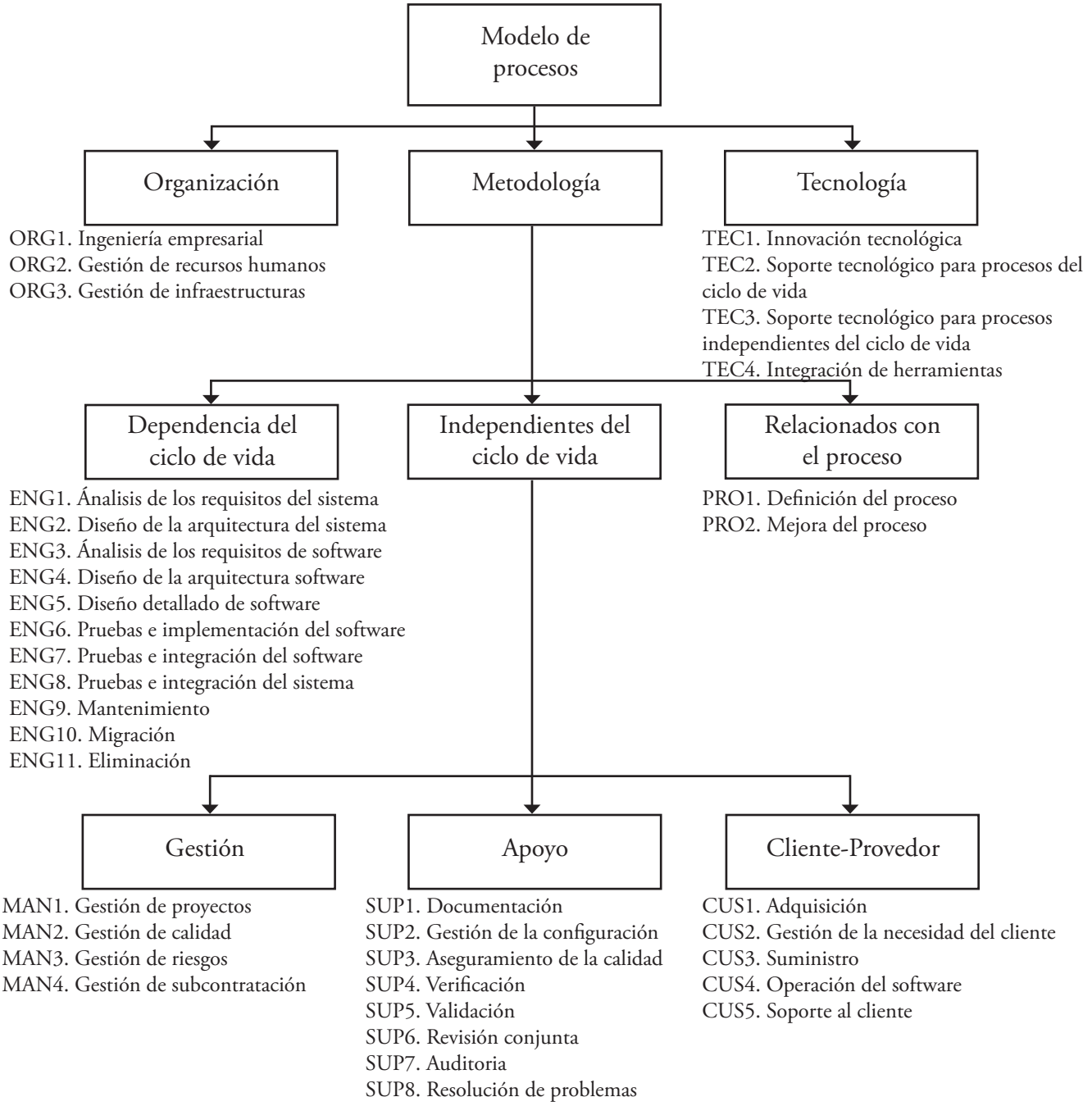


Figura 3.5. Arquitectura del modelo de procesos según BOOTSTRAP [Komi-Sirviö, 2004]

3.4.1. PROPÓSITO DE BOOTSTRAP

Es otra iniciativa para resolver la crisis del desarrollo software. Mediante el uso de prácticas, herramientas y estándares de calidad, esta metodología internacional mide, evalúa y propone mejoras al proceso de desarrollo de software. Su enfoque consiste en evaluar el proceso, y no el producto. Para esto realiza las siguientes tareas:

- Define las características de los procesos.
- Provee un análisis cuantitativo.
- Hace evidentes las fortalezas y debilidades.
- Identifica las áreas de mejora.
- Provee recomendaciones.
- Sugiere un plan de implementación.

Para cumplir con las tareas antes mencionadas BOOTSTRAP se compone de: un modelo, un proceso de evaluación, una base de datos de soporte, un proceso de mejora y de los instrumentos de evaluación.

3.4.2. VENTAJAS DE BOOTSTRAP

- Proporcionar soporte para la evaluación de la capacidad de los procesos utilizando un conjunto de prácticas de Ingeniería del Software.
- Incluye estándares de Ingeniería del Software reconocidos internacionalmente como fuentes para la identificación de las prácticas a considerar.
- Da soporte a la evaluación, indicando cómo el estándar de referencia ha sido aplicado en la organización evaluada.
- Identifica las fortalezas y debilidades de los procesos de la organización evaluada.
- Da soporte a la creación y aplicación de un plan de mejora que genere resultados aceptables y fiables, de forma que las acciones del plan de mejora permitan alcanzar los objetivos de la organización.

3.4.3. DESVENTAJAS DE BOOTSTRAP

- Existe poca documentación para el usuario y no existen herramientas que soporten su aplicación [Cabo & Moralejo, 2008].
- Problemas para ajustar el modelo de evaluación a las necesidades particulares del proyecto [Jarvinen, Hamann & Van Solingen, 1999]
- Suficiencia de recursos para soportar la implantación [Stienen, 1999].
- Genera documentación excesiva [Cabo & Moralejo, 2008].
- No existe evidencia de su adaptación para el entorno de SE.

3.5 MÉTODOS ÁGILES

Es fundamental que el software nuevo se desarrolle lo más rápido posible para aprovechar nuevas oportunidades y responder a la presión competitiva. Lo que provoca que el desarrollo y entrega rápidos sean a menudo los requisitos más críticos de los sistemas de software [Sommerville, 2010], haciendo necesario contar con procesos de desarrollo que produzcan software útil de forma rápida. Es por esto que en Febrero de 2001, tras una reunión celebrada en Utah, nació el término “ágil” aplicado al desarrollo de software.

En esta reunión participó un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y responder a los cambios que pudieran surgir a lo largo del proyecto.

Los métodos ágiles establecen que el software no sea desarrollado y utilizado en su totalidad sino en una serie de incrementos, por lo que propone el desarrollo del software en cortos lapsos de tiempo, minimizando posibles riesgos. Cada una de esas unidades de tiempo se denomina "iteración", la cual debe durar entre una y cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Cada iteración no debe añadir demasiada funcionalidad, para justificar el lanzamiento del producto al mercado, sino que la meta debe ser conseguir una versión funcional sin errores. Al final de cada iteración, el equipo volverá a evaluar las prioridades del proyecto. Algunas de las principales características de los métodos ágiles son las siguientes:

- La prioridad es satisfacer al cliente mediante entregas de software tempranas y continuas que le aporten un valor.
- Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- Entregar frecuentemente software que funcione desde un par de semanas hasta un par de meses, con el menor intervalo de tiempo posible entre entregas.
- La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
- Construir el proyecto alrededor de individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para finalizar el trabajo.
- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- El software que funciona es la medida principal de progreso.
- Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se organizan por sí mismos.
- En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

De acuerdo a [Sommerville, 2010], existen muchos enfoques para el desarrollo rápido de software que se fundamentan en las siguientes características:

- Los procesos de especificación, diseño e implementación son concurrentes. No existe una especificación detallada del sistema, el documento de requisitos del usuario define solamente las características más importantes del sistema. La documentación del diseño es minimizada o generada automáticamente por el entorno de programación.
- El sistema se desarrolla en una serie de incrementos. En cada incremento se pueden proponer cambios en el software o nuevos requisitos.

- A menudo se desarrollan las interfaces de usuario del sistema utilizando un sistema de desarrollo interactivo que permita un diseño rápido.

Las metodologías ágiles están mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Tales metodologías se enfocan en la gente y en los resultados. Probablemente el método más conocido es la Programación Extrema (*eXtreme Programming*, XP), que a continuación se describe. Sin embargo, otros enfoques ágiles son: SCRUM, Cristal, desarrollo de software adaptable, entre otros. Así, las metodologías ágiles están revolucionando la manera de producir software.

3.5.1. PROGRAMACIÓN EXTREMA

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito del desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. El nombre fue acuñado por Beck debido a que el enfoque fue desarrollado utilizando buenas prácticas reconocidas, como el desarrollo iterativo, y con la participación del cliente en niveles "extremos". En XP, todos los requisitos se expresan como escenarios (llamados historias de usuario), los cuales se implementan directamente como una serie de tareas. Los programadores trabajan en parejas y desarrollan pruebas para cada tarea antes de escribir el código. Todas las pruebas se deben ejecutar satisfactoriamente cuando el código nuevo se integra al sistema [Sommerville, 2005].

XP involucra varias prácticas (véase Tabla 3.5), las cuales han sido reconocidas por la industria como mejores prácticas durante el paso de los años. Dichas prácticas son llevadas al extremo para obtener más que la suma de las partes. Estas prácticas hacen de XP, un método ágil, especialmente adecuado para proyectos con requisitos imprecisos y muy cambiantes y donde existe un alto riesgo técnico. Es importante mencionar que las herramientas más importantes de soporte a XP están bajo alguna licencia libre y que, al basarse en el recurso humano, clasifica a las personas en las siguientes categorías: cliente, programador, encargado de pruebas (tester), encargado de seguimiento (tracker), entrenador, consultor, gestor.

3.5.2. PROPÓSITO DE LOS MÉTODOS ÁGILES

Los métodos ágiles pretenden ser una alternativa a los procesos de desarrollo tradicionales caracterizados por su total rigidez centrada en el control del proceso, actividades, artefactos, y en las herramientas y notaciones a usar. Los métodos ágiles son métodos de desarrollo iterativo que se centran en la especificación, diseño e implementación del sistema forma incremental y que implican directamente a los usuarios en el proceso de desarrollo [Sommerville, 2010]. Todo esto con el propósito de reducir los tiempos de ciclo y proporcionar mayor valor a todos los actores clave involucrados en el desarrollo de software [Srinivasan, Dobrin & Lundqvist, 2009]. De acuerdo con [Anacleto, 2005], el diseño ágil busca: mantener el código tan claro y simple como sea posible; realizar mejoras en cualquier momento; establecer la forma de comunicar el diseño a la gente que necesita entenderlo, usando código, diagramas y, sobre todo, la conversación.

3.5.3. VENTAJAS DE LOS MÉTODOS ÁGILES

Los métodos ágiles se han adoptado para el desarrollo de SE dado que se centran en la adaptabilidad y el trabajo en equipo, lo cual resulta útil en el codiseño de hardware y software. Algunas de las particularidades que hacen adecuadas las metodologías ágiles para desarrollar SE, son las siguientes:

- El software de aplicación para SE grandes requiere de comunicación exhaustiva y mecanismos de verificación.

Tabla 3.5. Prácticas de la programación extrema [Sommerville, 2010]

Principio o práctica	Descripción
Planificación incremental	Los requisitos se registran en tarjetas de historias y las historias a incluir en una entrega se determinan según el tiempo disponible y su prioridad relativa. Los desarrolladores dividen estas historias en tareas de desarrollo.
Entregas pequeñas	Se desarrolla primero el mínimo conjunto útil de funcionalidad que proporcione valor de negocio. Las entregas del sistema son frecuentes e incrementalmente añaden funcionalidad a la primera entrega.
Diseño sencillo	Sólo se lleva a cabo el diseño necesario para cumplir los requisitos actuales.
Desarrollo previamente probado	Se utiliza un sistema de pruebas unitarias automatizado para escribir pruebas para nuevas funcionalidades antes de que éstas se implementen.
Refactorización	Se espera que todos los desarrolladores refactoricen el código continuamente tan pronto como encuentren posibles mejoras. Esto conserva el código sencillo y mantenible.
Programación en parejas	Los trabajadores trabajan en pareja, verificando cada uno el trabajo del otro y proporcionando la ayuda necesaria para hacer siempre un buen trabajo.
Propiedad colectiva	Las parejas de desarrolladores trabajan en todas las áreas del sistema, de modo que no desarrollen islas de conocimiento y todos los desarrolladores posean todo el código. Cualquiera puede cambiar cualquier cosa.
Integración continua	En cuanto acaba el trabajo en una tarea, se integran en el sistema entero. Después de la integración, se deben pasar al sistema todas las pruebas de unidad.
Ritmo sostenible	No se consideran aceptables grandes cantidades de horas extras, ya que a menudo el efecto que tienen es reducir la calidad del código y la productividad a mediano plazo.
Cliente presente	Debe estar disponible al equipo de la XP un representante de los usuarios finales del sistema (el cliente) debe tener disponibilidad de tiempo completo para con el equipo. En un proceso de XP, el cliente es miembro del equipo de desarrollo y es el responsable de formular al equipo los requisitos del sistema para su implementación.

- El software es una parte importante del desarrollo de los SE, sin embargo es difícil de realizar debido a que estos sistemas forman parte de un entorno físico cuya dinámica compleja, además del calendario y los requisitos, debe ser satisfecha. Esto puede ser solventado con las metodologías ágiles pues ayudan a construir aplicaciones rápidamente en entornos en los que los requisitos cambian con frecuencia basados en la idea de que "*por la alta frecuencia en el cambio que sufren los requisitos, se requiere una menor necesidad de diseño y planificación inicial y mayor necesidad de desarrollos incrementales e iterativos*", así las metodologías ágiles se centran en desarrollar software funcional.
- El desarrollo del hardware de los SE se realiza mediante lenguajes de descripción hardware (VHDL, Verilog, etc), así como la verificación del hardware (OpenVera) y el modelado (SystemC) que ayudan a la obtención de prototipos rápidos.
- Ciclos de desarrollo cortos, los metodos ágiles tienen como propósito la realimentación rápida [Abrahamsson, 2005].
- Permiten a los desarrolladores centrarse en el software mismo en vez de en su diseño y documentación [Sommerville, 2005].
- En el artículo [Srinivasan, Dobrin& Lundqvist, 2009], se manifiesta el uso de XP en el desarrollo de SE a través de un estudio sobre el estado actual de la práctica, así como algunas directrices entre las que se pueden resumir las siguientes:
    - Los métodos ágiles requieren de la infraestrucutura adecuada, incluyendo las herramientas adecuadas para apoyar el ciclo de vida del software, así como la creación de un entorno que fomente la comunicación y colaboración.
    - Los métodos ágiles requieren del intercambio y la transferencia de conocimiento para su adopción exitosa.

3.5.4. DESVENTAJAS DE LOS MÉTODOS ÁGILES

- Falta de documentación del diseño.
- Fuerte dependencia de las personas.
- Falta de procesos de revisión del código.

3.6 COMPARACIÓN EMPÍRICA SOBRE LOS MODELOS Y ESTÁNDARES DE REFERENCIA PARA MEJORAR LA CALIDAD DE LOS PRODUCTOS DE DESARROLLO SOFTWARE

De los modelos y estándares presentados en los apartados anteriores se presenta la siguiente comparación empírica orientada al desarrollo de Sistemas Empotrados. Así, la Tabla 3.6 extrae las características más notables de los principales modelos y estándares para la mejora de la calidad de desarrollo de software. La comparación empírica tiene como objetivo conocer el alcance de los modelos al aplicarlos en el desarrollo de los SE, así como conocer las ventajas y desventajas que presentan su uso. Los criterios de comparación de la Tabla 3.6 están basados en la revisión de la literatura, estudios comparativos existentes y en la cobertura de CMMI-DEV.

A continuación se describen los criterios más relevantes:

- **Ámbito de aplicación:** con este criterio se delimita el uso del modelo o estándar, es decir si su aplicación es generalizada, orientada a sistemas, a software o a SE.

- **Autor:** en esta casilla se anoto el nombre de la organización o investigador que desarrollo el modelo o estándar.
- **Objetivo:** se describe el fin, meta o finalidad que busca cumplir el modelo o estándar.
- **Puntos a favor:** describe las principales ventajas de usar el modelo o estándar en el ámbito de aplicación para el que fue desarrollado.
- **Puntos en contra:** muestra las desventajas que se presentan al aplicar el modelo o estándar en el ámbito de aplicación para el que fue desarrollado.
- **Representación:** Este criterio identifica la organización, uso y presentación de los componentes del modelo o estándar.
- **Costo de implementación:** determina si el modelo o estándar es costoso en su implementación y en la obtención de los niveles de madurez. Con este criterio se puede determinar el impacto de la aplicación del modelo o estándar en el costo del desarrollo de SE.
- **Facilidad de implementación:** este criterio está destinado a reflejar el nivel de dificultad que implica implementar el modelo o estándar.
- **Repositorio de activos:** Este criterio es vital para ir mejorando en el desarrollo de SE, pues es donde se almacenan resultados, análisis, riesgos, etc. del diseño y desarrollo de SE.
- **Plantillas y documentos propios:** en este campo se define si el modelo o estándar establece plantillas o documentos que formen una guía para el proceso de desarrollo de los SE, así como también facilite el aprendizaje para desarrollar SE de calidad.
- **Aplicado en SE:** para determinar si existen antecedentes de que el modelo o estándar ha sido empleado en el desarrollo de Sistemas Empotrados.

Tabla 3.6. Tabla comparativa de las metodologías actuales para el desarrollo de SE

Metodología	CMMI-DEV	ISO 15504	Six Sigma	BOOSTRAP	Métodos agiles
Ámbito de aplicación	Sistemas y software	Sistemas y software	Genérico	Software	Sistemas y software
Autor	SEI	ISO	Bill Smith	Kuvaja	Kent Beck
Objetivo	Ayudar a las empresas desarrolladoras de sistemas a mejorar sus procesos siguiendo un camino evolutivo que va desde un nivel inicial *ad hoc* hasta alcanzar la madurez.	Mejorar y evaluar los procesos de desarrollo y mantenimiento de sistemas y productos de software.	Desarrollar procesos, productos y/o servicios eficientes y minimizar los defectos asociados hasta un valor objetivo.	Medir, evaluar y proponer mejoras al proceso de desarrollo software.	Disminuir los tiempos de ciclo y proporcionar mayor valor a todos los actores clave involucrados en el desarrollo de software.
Puntos a favor	Es apto para el estudio, diseño, desarrollo, implementación y soporte de software de aplicación y hardware. Reconocido internacionalmente.	Es una norma internacional y es el primer modelo bidimensional, al separar los procesos y capacidad en dimensiones diferentes.	Reducción de los defectos del sistema. Evaluación de los procesos y métodos productivos. Cultura de mejora continua basada en la autorrevisión.	Evaluación y mejora de procesos de software. Incluye estándares de Ingeniería del Software.	Ciclos de desarrollo cortos. Se centrar en el software.

Tabla 3.6. Tabla comparativa de las metodologías actuales para el desarrollo de SE (continuación)

Metodología	CMMI-DEV	ISO 15504	Six Sigma	BOOSTRAP	Métodos agiles
Puntos en contra	Exige un alto esfuerzo para su implantación y una inversión alta para ser completamente implementado	Difícil en capacidad, complejo para evaluar, se llega a dar solapamiento de las dimensiones	No garantiza la calidad	Evalúa el proceso, no el producto.	Falta de documentación del diseño
Representación	Continua y por etapas	Continua (por etapas a nivel de proceso)	-	-	-
Costo de implementación	Alto (requiere personal calificado)	Alto	Alto (Inversión inicial en la formación del personal)	Media	Baja
Plantillas y documentos propios	No	No	No	No	No
Repositorio de activos	No	No	No	No	No
Fácil de implementar	No	No	No	No	Sí
Aplicado en SE	Si	No	No	Si	Si

4

INTRODUCCIÓN A LA SOLUCIÓN

De acuerdo a la exploración de la literatura presentada en el Capítulo anterior, es evidente que se han realizado muchos esfuerzos para establecer "qué actividades realizar" en el desarrollo de SE, en vez de "cómo realizarlas". Por lo que, el problema actual con el desarrollo de los SE no es la falta de normas o modelos, sino la falta de una estrategia eficaz para aplicarlos con éxito. Por lo que resulta evidente la necesidad de optimizar el proceso de desarrollo de los SE para hacer frente a los retos de la puntualidad, productividad y calidad que exige.

A continuación se retomará los resultados del análisis de las metodologías de diseño presentadas en el Capítulo 3 con el objetivo de introducir la metodología alternativa para el diseño de SE, propuesta en este trabajo de tesis.

4.1 PROPUESTA FORMAL DE SOLUCIÓN

Mejorar los métodos convencionales utilizados para el desarrollo de los SE en entornos industriales y académicos es una preocupación latente que ha propiciado el desarrollo de muchas propuestas para incorporar la formalidad a este "proceso" y que, como ya se mencionó anteriormente, éstas no consideran las necesidades específicas del desarrollo de este tipo de sistemas. Como resultado de esta práctica, los métodos, herramientas y técnicas usadas para el desarrollo de SE no solo varían entre compañías e institutos de investigación, sino también dentro de las mismas empresas y universidades, así, los desarrolladores de SE no tienden a desarrollar productos con estandarizaciones, elemento crucial para mejorar la calidad del producto final. A partir de la investigación realizada en los Capítulos 2 y 3, se compararán las metodologías o métodos en base a las fases, herramientas, buenas práctica y forma en que abordan los problemas del desarrollo de los SE. Es importante mencionar que se observó que las propuestas actuales para el desarrollo de SE se basan principalmente en tres líneas: *herramientas*, *componentes* y *mejora al proceso software.*

- ***Componentes***: La mejora de los productos empotrados enfocó sus esfuerzos a la reutilización de componentes, para mejorar el proceso de desarrollo e impactar en la calidad del proceso de desarrollo de SE. La reutilización de componentes permite reutilizar piezas de código pre-elaborado. En la literatura se encontraron varios casos que integran la reutilización de componentes para el desarrollo de SE (p.ej., ESCM [Li et al., 2009] y Save-IDE [Sentilles et al., 2009]). En resumen, el modelo ESCM se desarrolló para la especificación, la verificación y la composición de software empotrado basado en componentes. ESCM describe cómo especificar los componentes desde cuatro perspectivas: sintáctica, funcional, de calidad del servicio (QoS) y de sincronización. En la misma línea se encuentra Save-IDE, cuya contribución principal está relacionada con el esfuerzo para establecer un proceso de desarrollo software, señalando como fases principales: el diseño, el análisis y la realización. Entre las ventajas de usar el enfoque del diseño basado en componentes en SE, se encuentran: la reutilización de componentes que conlleva a mejorar la calidad de los SE, la reducción del ciclo de desarrollo y un mayor retorno sobre la inversión. La Tabla 4.1. resume las características evaluadas en ESCM y Save-IDE.

Tabla 4.1. Aspectos cubiertos por los modelos o metodologías orientadas al uso de componentes

	ESCM	Save-ID
Separación en fases	Sí	Sí
Administración de requisitos	No se especifica	No se especifica
División de componentes	Sí	Sí
Modelado	No se especifica	No se especifica
Reutilización de componentes	Sí	Sí
Repositorio de componentes	No se especifica	No se especifica
Artefactos	No se especifica	No se especifica

- ***Herramientas:*** Se han desarrollado metodologías para el diseño e implementación de SE tal como REMES [Seceleanu, Vulgarakis & Petterson, 2009], ROPES [Powel, 2004] y ModES [Riccobene et al., 2006] basadas en el uso de herramientas.

En esta línea, el modelo REMES está basado en una máquina de estados con soporte para el modelado jerárquico, anotaciones de recursos, tiempo continuo, y nociones de puntos explícitos de entrada y salida, que resultan convenientes para el modelado de SE basado en componentes. Por otra parte, ROPES se centra en el uso de UML y ModES es una metodología y un conjunto de herramientas que aplican el enfoque MDE para el diseño de SE. En la Tabla 4.2 se presentan los aspectos cubiertos por REMES, ROPES y ModES.

Tabla 4.2. Aspectos cubiertos por los modelos o metodologías orientadas al uso de herramientas

	REMES	ROPES	ModES
Separación en fases	No se especifica	Si	No se especifica
Administración de requisitos	Si	Sí	No se especifica
División de componentes	No	No se especifica	No se especifica
Modelado	No se especifica	Si	Si
Reutilización de componentes	No se especifica	No se especifica	No se especifica
Repositorio de componentes	No se especifica	No se especifica	Si
Artefactos	No se especifica	Sí	Si

- ***Mejora al Proceso Software:*** Los métodos desarrollados bajo el enfoque SPI parecen ser adecuados para el desarrollo de SE, pero su uso no se ha convertido en una práctica organizada. Tal como sucede con el desarrollo de software, el desarrollo de los SE podría fallar a causa de una mala gestión de todo el proceso de desarrollo. CMMI es un modelo que se ha usado en el campo de los SE por la gran eficacia y por su estructura para la industria de las TI, lo que lo hace apto para el estudio, diseño, desarrollo, implementación y soporte de software de aplicación y hardware. Los trabajos realizados en [Jun, Rui & Yi-min, 2007] y [Solingen, 2002] proporcionan una solución orientada a SPI relacionada con el modelo CMMI en sus Niveles 2 y 3. Las investigaciones intentan proveer especificaciones de desarrollo técnico para SE. SPP, proporciona una solución orientada a la Mejora del Proceso Software relacionada con el modelo CMMI en sus Niveles 2 y 3 y enfocada a empresas medianas.

Tabla 4.3. Aspectos cubiertos por los modelos o metodologías orientadas a SPI

	SPP	*Product Focused Software Process Improvement*
Separación en fases	Si	Si (Se organiza por capas)
Administración de requisitos	Si	Si
División de componentes	No	No
Modelado	No	No
Reutilización de componentes	No	No
Repositorio de componentes	No	Si
Artefactos	No	No

En base al estudio realizado, resulta conveniente integrar las siguientes características a una metodología alternativa:

- Separación en fases: para el desarrollo de una metodología se detectó que es importante dividir en fases el desarrollo del sistema para controlar el progreso del proyecto y proporcionar periodos planificados de evaluación y de toma de decisiones. Por lo que, la metodología se definirá por fases que se componen de actividades específicas.
- Administración de requisitos: con la administración de requisitos se obtiene una aproximación sistemática para la búsqueda, documentación, organización y seguimiento en los cambios en los requisitos de un sistema. Para los SE es importante realizar una adecuada administración para integrar y coordinar los requisitos del software, del hardware y demás partes que integran el sistema. Por otra parte, los requisitos de los SE son "requisitos especiales".
- División de componentes: con la división de componentes se maneja la complejidad y se tiene un mejor control sobre los posibles riesgos del proyecto. Además de mejorar los factores de calidad del software, facilitar la reutilización de código, entre otras cosas. La división de componentes identifica la arquitectura del sistema y las relaciones entre los componentes.
- Modelado: la arquitectura del sistema debe modelarse como un todo, partiendo de la idea de los subsistemas que a su vez deben ser modelados. Una vez que se tiene la arquitectura del sistema, las porciones del mismo pueden ser manejadas por los equipos de desarrollo de acuerdo a su disciplina. La organización del modelo generalmente no es considerada hasta que el equipo de desarrollo se topa con algún problema.
- Desarrollo del software de forma iterativa: se planifican incrementos basados en las prioridades del usuario y del desarrollo.
- Artefactos: la mayoría de metodologías analizadas utilizan un repositorio de documentos que controla ciertas tareas específicas. El añadir un repositorio más amplio incrementaría la gestión en el desarrollo de los SE, además de servir como un elemento de guía y/o ayuda en la realización de las actividades.

Así, la metodología propuesta, integrará al CMMI-DEV v1.2 [CMMI, 2006] en su Nivel 2, para especificar el proceso a seguir para desarrollar SE de calidad. Además, la metodología añade los principios del *Team Software Process* [Humphrey, 2000] para incorporar la capacidad de gestión y mejorar así la especificación del proceso; además de que permite establecer un conjunto de documentos para guiar a los desarrolladores en la gestión del proyecto. Esta metodología será denominada de aquí en adelante como SPIES (*Software Process Improvement for Embedded Systems*).

4.2 PLANTEAMIENTO FORMAL DE LA HIPÓTESIS

La hipótesis del presente trabajo de tesis (presentada en el Capítulo 1) se soporta de mejor manera como sigue: *Usando CMMI-DEV v1.2 como modelo de referencia, es posible desarrollar una metodología para el desarrollo de Sistemas Empotrados bajo el paradigma de Mejora del Proceso Software; que planifique, administre y controle el desarrollo de estos sistemas. La metodología guiará a los desarrolladores durante todo el ciclo de vida de los SE a través de la definición de las fases de los procesos, actividades y subactividades; utilizando el modelo TSP para indicar qué usar, cómo usarlo y cuándo usarlo, con un énfasis en la mejora continua. De tal forma que el desarrollo de SE se planificará, administrará y controlará.*

4.3 ESTRUCTURA DE LA METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMPOTRADOS BAJO EL ENFOQUE DE MEJORA DEL PROCESO SOFTWARE

Una metodología consiste de un lenguaje para especificar los elementos y las relaciones entre los componentes de un sistema, y un proceso completo (actividades, productos, entradas, salidas, métricas, criterios de entrada, criterios de salida, roles y más) que indique qué partes del lenguaje utilizar, cómo utilizarlas y cuándo utilizarlas [Powel, 2006]. Bajo este concepto se diseño *Software Process Improvement for Embedded Systems* (SPIES). SPIES tiene como objetivo mejorar la calidad del software y la productividad de desarrollo de los SE mediante el fortalecimiento de su diseño, empleando técnicas de modelado. A continuación se describirán los componentes de la solución propuesta (SPIES), y se utilizarán estos elementos para responder a las hipótesis planteadas en el Capítulo 1 y que serán retomadas más adelante.

4.3.1 COMPONENTES DE LA METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMPOTRADOS BAJO EL ENFOQUE DE MEJORA DEL PROCESO SOFTWARE

SPIES es la metodología propuesta para el desarrollo de SE y especifica un conjunto integrado de actividades para guiar a los desarrolladores durante la creación de sistemas robustos, capaces y seguros. SPIES integra al CMMI-DEV v1.2 en su Nivel 2, para especificar el proceso a seguir para desarrollar SE de calidad; consta de tres elementos esenciales: actividades, activos y herramientas.

Esta metodología incorpora los principios del *Team Software Process* para incorporar la capacidad de gestión y mejorar así la especificación del proceso. La metodología está estructurada por ocho fases con 16 áreas de proceso que se componen de actividades más específicas. SPIES ha sido diseñada con un enfoque iterativo para asegurar que el proceso de desarrollo sea probado en cada fase y no hasta el final del proyecto. Por otra parte, ilustra los formatos de diferentes documentos mediante plantillas (cerca de 12 plantillas), e indica cómo cumplirlas ademas de proporcionar guías que especifican cómo realizar y adaptar la especificación y las fases. Por lo anterior se ha incorporado un repositorio de conocimiento a cada fase para asegurar altos niveles de éxito.

SPIES utiliza UML como lenguaje de modelado[7] , porque los sistemas empotrados exigen un nuevo enfoque; cada fase de la metodología puede ser representada con un diagrama de actividades de UML. Esta metodología conceptual incluye explícitamente a la medición y la aplica en estos dos propósitos: evaluar al sistema y al proceso. Así, cada desarrollador sabe cuándo realizar una actividad de acuerdo a la especificación de SPIES. La Figura 4.1 presenta una explicación gráfica sobre las relaciones de las áreas de proceso que componen a SPIES.

Las áreas de proceso en la metodología son un grupo de prácticas relacionadas que, cuando se implementan colectivamente, satisfacen un conjunto de metas consideradas importantes para hacer mejoras en esa área. A continuación se describen los procesos de la metodología SPIES:

- ***Planeación del proyecto.*** Este proceso proporciona la guía para establecer y mantener los planes que definen las actividades del proyecto. La planeación incluye la estimación de los atributos de cada producto y de las tareas; la determinación de los recursos necesarios, así como la identificación y análisis de los riesgos del proyecto, la programación de actividades, y las consideraciones para escoger el procesador correcto para el SE.

7 Para más información sobre UML consultar el Anexo C de esta tesis.

- ***Especificación de requisitos.*** Este proceso proporciona las actividades necesarias para el desarrollo y la administración de los requisitos. Dependiendo de los lenguajes de modelado utilizados para describir los requisitos, los desarrolladores pueden utilizar lenguajes gráficos, tales como UML o el Lenguaje de Modelado de Sistemas (SysML).
- ***Diseño del producto.*** Este proceso proporciona las actividades para crear el diseño eficaz del sistema basado en los requisitos anteriores y enfocados en dos cuestiones: diseño funcional y diseño de la arquitectura. SPIES indica a los desarrolladores cómo definir la arquitectura del sistema en base a los requisitos y al diseño funcional.
- ***Desarrollo electromecánico.*** Este proceso proporciona las actividades que permiten que los desarrolladores administren y gestionen la información y los elementos para el diseño software, mecánico y eléctrico. SPIES recomienda las revisiones controladas automáticas para identificar problemas potenciales entre disciplinas y proporciona herramientas de colaboración para solucionar incidencias en el proceso.
- ***Diseño automatizado.*** Esta fase proporciona prácticas para integrar todos los componentes del sistema en un modelo unificado. El proceso considera los requisitos de terceros durante el diseño del SE y su software. Además, se realiza una simulación de todo el SE.
- ***Diseño electrónico.*** Este proceso incluye actividades para utilizar las herramientas para diseñar y producir sistemas electrónicos que van desde placas de circuitos impresos (PCB) a los circuitos integrados.
- ***Desarrollo de software.*** Este proceso contiene las actividades para refinar o para extender modelos de la plataforma-independiente en el diseño plataforma-específica.
- ***Otras tecnologías de desarrollo.*** Es posible que el SE utilice otras tecnologías que afecten a la funcionalidad prevista. Por lo que la metodología ofrece un proceso con prácticas para desarrollar SE sin dependencias tecnológicas.
- ***Integración.*** Este proceso proporciona las actividades para integrar los componentes generados en los pasos anteriores. La fase de integración del ciclo de desarrollo debe contar con herramientas y métodos especiales para manejar la complejidad del proceso. El proceso de integración de software empotrado y el hardware es un ejercicio en depuración y descubrimiento.
- ***Validación.*** En esta fase se incluyen pruebas antes de que el sistema pase a la fase de entrega y mantenimiento.
- ***Entrega y mantenimiento.*** La mayoría de los diseñadores de SE (alrededor del 60%) mantiene y actualiza los productos existentes, en lugar de diseñar nuevos productos. Los desarrolladores deben utilizar la documentación existente y entender el diseño original lo suficiente como para mantener y mejorar. Este proceso requiere de herramientas que se adapten a la reingeniería y establezcan el escenario actual del SE.
- ***Medición y análisis.*** El proceso de medición es un requisito previo para todos los procesos anteriores y para lograr mejoras de proceso exitosas. En este contexto, la medición de SPIES debe usarse para dos fines: evaluación conforme a las especificaciones de calidad del SE, y evaluación de la relación proceso-sistema.

Es importante mencionar que el contenido de los procesos de SPIES no puede ser modificado, mientras que la modificación/reutilización de los activos anteriores de los proyectos (como documentos, plantillas, pautas, valoraciones, etc.) está permitido. SPIES se compone de tres categorías de aplicación (véase Figura 4.2). Las categorías son dependientes y se relacionan para la definición de un proceso efectivo.

En SPIES, las áreas de proceso son descritas en función de su propósito, actividades por objetivo, actividades específicas, notas introductorias y áreas de proceso relacionadas; a continuación cada componente será descrito de forma detallada.

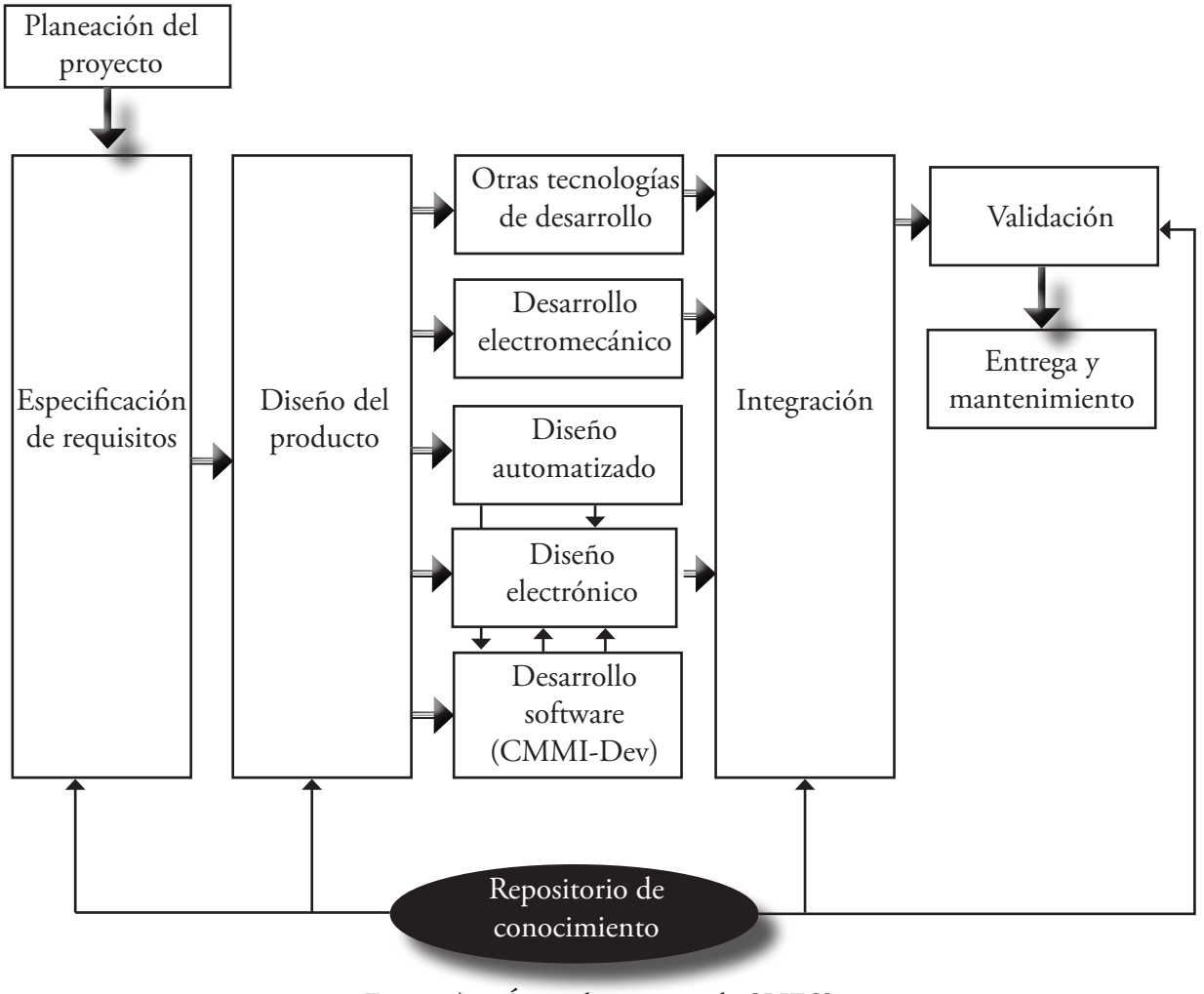


Figura 4.1. Áreas de proceso de SPIES[8]

8 El proceso MAN no se muestra porque sus actividades están implícitas en los otros 11 procesos

Áreas de proceso: se refiere a un grupo de prácticas relacionadas en un área que, cuando se implementan colectivamente, satisfacen un conjunto de metas consideradas importantes para hacer mejoras en esa área.

Propósito: en esta sección se describe el objetivo del área de proceso. De forma breve se plasma la intención de implementar el área de proceso para mejorar el desarrollo de SE.

Notas introductorias: es un mensaje informativo y breve acerca del área de proceso. Las "notas introductorias" tienen la finalidad de introducir al lector al área de proceso relacionada mediante una descripción de ésta, lo que implica su realización y la importancia para el diseño de SE.

Áreas relacionadas: Esta sección es un componente informativo y lista las referencias a áreas de proceso que están en relación y refleja las relaciones de alto nivel entre las áreas de proceso.

Resumen de objetivos: Esta sección trata de forma breve y precisa acerca de lo que se pretende alcanzar como resultado de las actividades propuestas por SPIES en el área de proceso actual.

Actividades por objetivo: Consiste en un conjunto de operaciones o tareas propias. Las acciones del proyecto durante su implementación. Las actividades resultan en productos.

Productos a obtener: La sección especifica los artefactos del desarrollo del área de proceso, tal como, modelos, código, documentación, planes de trabajo, plantillas, etc.

Actividades específicas: Es la descripción de las actividades que se consideran importantes para alcanzar los objetivos.

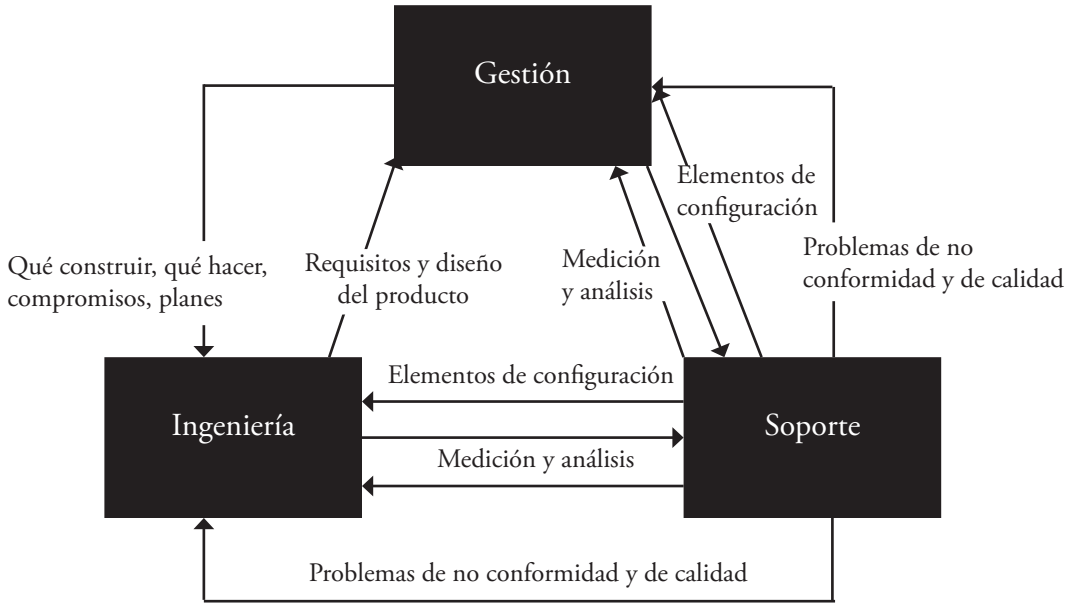


Figura 4.2. Categorías de procesos de SPIES

5

METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMPOTRADOS BAJO EL ENFOQUE DE MEJORA DEL PROCESO SOFTWARE

En este capítulo se presenta la metodología SPIES, que integra al CMMI-DEV v1.2 en su Nivel 2, para especificar el proceso a seguir para desarrollar Sistemas Empotrados de calidad. Se describen las ocho fases, 16 áreas de proceso y las 12 plantillas que integran la metodología.

5.1. PLANIFICACIÓN

Un área de proceso de la categoría de Gestión de SPIES

PROPÓSITO

El propósito de la Planificación consiste en establecer y mantener planes de trabajo que definan las actividades del proyecto.

NOTAS INTRODUCTORIAS AL ÁREA DE PROCESO

La planificación del proyecto en SPIES involucra lo siguiente:

- Desarrollar el plan del proyecto.
- Gestionar la relación con las personas involucradas de forma adecuada.
- Interactuar apropiadamente con los proveedores de requisitos.
- Mantener el plan.

La planificación inicia con los requisitos que definen al sistema empotrado y al proyecto como tal. Regularmente, cuando se trata de un sistema con finalidad escolar (ya sea práctica de laboratorio o desarrollo de tesis) los requisitos están relacionados directamente con la funcionalidad del sistema. En esta parte de la planificación esta funcionalidad debe ser traducida a un enunciado específico en donde se deben plasmar qué usará el sistema como entrada, cuál será el resultado que entregará, cómo se hará el procesamiento de esa entrada para obtener dicha salida, y parámetros relacionados con lo anterior (entradas, salidas y procesamiento). La planificación incluye la estimación de aspectos relacionados con los productos y las tareas a desarrollar, la determinación de los recursos necesarios para construir el sistema, la negociación de los compromisos, la elaboración de un calendario, y la identificación y análisis de los riesgos del proyecto. El plan del proyecto necesitará ser revisado de acuerdo al avance del proyecto para tratar los cambios en los requisitos y compromisos, las estimaciones inexactas, las acciones correctivas, y los cambios del proceso. El término *"plan de proyecto"* es usado durante todas las actividades y áreas de SPIES para referirse al plan general que controla el proyecto.

ÁREAS DE SPIES RELACIONADAS

Consultar el área de Especificación de Requisitos para mayor información sobre la definición y desarrollo de los requisitos que definen el producto y componentes del producto. Estos requisitos (de producto y componentes) sirven como base para la planificación y replanificación.

RESUMEN DE OBJETIVOS:

1. Establecer estimaciones.
    - 1.1. Estimar el alcance del proyecto.
    - 1.2. Establecer estimaciones para productos y tareas.
    - 1.3. Determinar estimaciones de costo y esfuerzo.
2. Desarrollar el plan del proyecto.
    - 2.1. Establecer presupuesto y calendario.
    - 2.2. Identificar los riesgos del proyecto.

2.3. Planificar la necesidad de conocimientos.
2.4. Establecer el plan del proyecto.

ACTIVIDADES POR OBJETIVO

5.1.1. ESTABLECER ESTIMACIONES

Se establecen las estimaciones de los parámetros de la planificación del proyecto

Los parámetros de la planificación del proyecto incluyen toda la información necesaria para que el proyecto se realice, su organización, la asignación de personal, la coordinación, divulgación y presupuestos necesarios. Las estimaciones de los parámetros de la planificación deben tener una base para inspirar la confianza de que cualquier plan basado en estas estimaciones será capaz de soportar los objetivos del proyecto. Los factores que regularmente son considerados cuando se estiman estos parámetros son los siguientes:

- Los requisitos del proyecto, incluyendo los requisitos del producto, los requisitos impuestos por el desarrollador, los requisitos impuestos por el cliente, y otros requisitos que afectan al proyecto.
- El alcance del proyecto.
- Las tareas y productos identificados.
- Enfoque técnico.
- La estrategia del proyecto.
- Atributos de los productos y tareas (tamaño o complejidad).
- Calendario.
- Modelos o datos históricos para convertir los atributos de los productos y tareas en costo y horas de trabajo.

La documentación de la estimación realizada es necesaria para la revisión de los participantes en el proyecto y para el mantenimiento del plan cuando el proyecto avance.

5.1.1.1. Estimar el alcance del proyecto

Establecer estimaciones para las actividades del plan del proyecto

El desarrollo de las actividades propuestas por SPIES con un ciclo iterativo permite dividir a todo el proyecto en un conjunto interconectado de componentes manejables. Regularmente, cualquier conjunto ordenado de actividades es una estructura orientada al proceso que proporciona un esquema para identificar y organizar las unidades lógicas de trabajo a ser gestionadas. La estructura de trabajo de SPIES proporciona una referencia y un mecanismo organizacional para la asignación del esfuerzo, calendario, y responsabilidad y es usado como el marco subyacente para planificar, organizar y controlar el trabajo realizado en el proyecto.

PRODUCTOS A OBTENER:

1. Descripciones de las tareas.
2. Descripciones de los resultados para cada tarea.
3. Estructura formal de actividades.

Actividades específicas

1. Analizar las actividades del plan de proyecto de SPIES (Artefacto PRY) para entenderlas y comprender el desarrollo del mismo

La estructura de actividades aportada por SPIES proporciona el esquema para organizar el trabajo del proyecto alrededor del producto y sus componentes. Esta estructura de actividades debe permitir una primera visión sobre los siguientes elementos:

- Posibles riesgos por actividad y tareas de mitigación.
- Identificar tareas para entregables y soporte.
- Tareas para desarrollo de planes adicionales, tales como la Gestión de Configuración y Aseguramiento de la Calidad del Proceso y Producto y Validación.

2. Identificar tiempos estimados para las tareas del plan, y asegurar la asignación de responsabilidades y el calendario del proyecto.

El objetivo de las actividades en el plan de SPIES es ayudar a que los desarrolladores establezcan tiempos iniciales de esfuerzo sobre el proyecto para terminar las tareas y mejorar la visión general de la asignación de roles, responsabilidades y tareas en el proyecto. La cantidad de detalle especificada ayuda en el desarrollo de calendarios realistas, reduciendo al mínimo la necesidad de gestión.

3. Identificar el producto y componentes del producto que serán desarrollados externamente (Artefacto STR).

4. Identificar los componentes que serán reutilizados (Artefacto STR).

5.1.1.2. Establecer actividades para productos y tareas

Establecer estimaciones para los productos y las tareas del proyecto

Para muchos modelos de estimación, el tamaño es la entrada principal para determinar el esfuerzo, costo y calendario. Sin embargo, los modelos también se pueden basar en entradas como la complejidad y estructura de las tareas.

Dentro de los ejemplos de productos para los que se realizan estimaciones de tamaños se encuentran los siguientes:

- Productos entregables y no entregables
- Documentos y archivos
- Hardware, firmware y software

SPIES recomienda los siguientes tipos de medidas de tamaño:

- Número de funciones (programa)
- Puntos función (programa)
- Líneas de código fuente (programa)
- Número de clases y objetos (diseño)
- Número de requisitos (especificación de requisitos)
- Número de interfaces (diseño)
- Número de páginas (documentos)
- Número de entradas y salidas (procesos)
- Número de puertas lógicas para los sistemas integrados
- Número de partes (tarjetas de circuito impreso, componentes, y partes mecánicas)
- Restricciones físicas (peso y volumen)

PRODUCTOS A OBTENER:

1. Estrategia de desarrollo.
2. Tamaño y complejidad de las tareas y productos.
3. Estimaciones de los parámetros.

Actividades específicas

1. Determinar la estrategia de desarrollo para el proyecto (Artefacto STR)

La estrategia define una guía para el desarrollo del producto. Esta incluye las decisiones sobre las características de la arquitectura, tales como un desarrollo distribuido o cliente/servidor; estado del arte o tecnologías establecidas para ser aplicadas, tales como robótica, materiales compuestos, o inteligencia artificial; y la amplitud de la funcionalidad esperada en los productos finales, como la seguridad, confianza y ergonomía.

2. Utilizar métodos apropiados para determinar los parámetros de los productos y tareas que serán usados para estimar el tiempo y esfuerzo necesario del proyecto.

Los métodos para determinar el tamaño y complejidad del sistema deben estar basados en modelos validados o datos históricos de proyectos anteriores.

Dentro de los ejemplos de métodos actuales se incluyen los siguientes:

- Número de puertas lógicas para el diseño de circuitos integrados
- Líneas de código o puntos función para software
- Número/complejidad de los requisitos para la ingeniería de sistemas

3. Estimar los atributos de los productos (Artefacto TAM) y tareas (Artefacto PRY)

5.1.1.3. Determinar estimaciones de costo y esfuerzo

Estimar el esfuerzo y costo del proyecto para los productos y tareas basándose en estimaciones realistas

Las estimaciones de costo y esfuerzo generalmente están basadas en el análisis utilizando modelos o datos históricos aplicados al tamaño, actividades y otros parámetros de la planificación. La confianza en estas estimaciones está basada en la razón fundamental del modelo seleccionado y la naturaleza de los datos. Pueden existir ocasiones cuando los datos históricos no puedan ser utilizados, por ejemplo cuando los esfuerzos no tienen precedentes o donde el tipo de tarea no encaja con los modelos disponibles. Un esfuerzo no tiene precedente (hasta cierto punto) si nunca ha sido desarrollado un producto o componente similar. Un esfuerzo puede también no tener precedente si el grupo de desarrollo (o el desarrollador) nunca ha desarrollado un producto o componente similar. Los esfuerzos sin precedente tienen mayor riesgo, requieren de más investigación para desarrollar estimaciones razonables, y requieren mayor reserva de gestión. La singularidad del proyecto debe ser documentada cuando se utilizan esos modelos para asegurar el entendimiento común de cualquier suposición hecha en las etapas iniciales de la planificación. Por otra parte, si se cuentan con datos históricos y se combina con un proceso de estimación adecuadamente refinado y ajustado al proyecto se puede realizar una estimación medianamente fiable. SPIES utiliza un modelo de estimación basado en horas/esfuerzo para esta actividad.

PRODUCTOS A OBTENER:

1. Estimación.
2. Estimaciones de esfuerzo del proyecto.
3. Estimaciones de costo del proyecto.

Actividades específicas

1. Recoger los datos históricos que serán usados para transformar los parámetros de los productos y tareas en estimaciones de horas/trabajo y costo.

Muchos modelos paramétricos han sido desarrollados para ayudar en la estimación del coste y calendario. No se recomienda el uso de estos modelos como fuente única de estimación, porque están basados en datos históricos del proyecto que pueden o no ser pertinentes a otro tipo de proyecto. Pueden utilizarse múltiples modelos y/o métodos para asegurar un alto nivel de confianza en la estimación.

Los datos históricos incluyen datos del costo, esfuerzo y calendario de proyectos ejecutados previamente, más datos de escalas apropiadas para dar cuenta de diferentes tamaños y complejidades. Al inicio, se recomienda probar con estimaciones basadas en la experiencia del desarrollador para aprender según avanza el desarrollo del proyecto y generar así datos históricos para proyectos futuros.

2. Incluir las necesidades de infraestructura de apoyo al estimar esfuerzo y costo.

La infraestructura de apoyo incluye la necesidad de recursos desde una perspectiva de desarrollo y sustento del producto. Se debe considerar las necesidades de recursos de infraestructura del entorno de desarrollo, el entorno de prueba, el entorno de producción, el entorno objetivo, o cualquier combinación apropiada de éstos cuando se estime el esfuerzo y costo.

Dentro de los ejemplos de recursos de infraestructura se incluyen los siguientes:

- Recursos de ordenador críticos (memoria, capacidad en disco y en red, periféricos, canales de comunicación y su capacidad)
- Entornos y herramientas de ingeniería (herramientas para los prototipos, ensamblaje, Rhapsody, LabView, Modelica, herramientas CAD y de simulación)

3. Estimar costo y esfuerzo usando modelos y/o datos históricos.

Las entradas de esfuerzo y costo usadas para la estimación son:

- Las estimaciones a juicio proporcionadas por un experto o grupo de expertos.
- Riesgos, incluyendo la extensión para la cual no existe un precedente de esfuerzo.
- Capacidades y roles críticos necesarios para realizar el proyecto.
- Requisitos del producto y sus componentes.
- Estrategia de desarrollo.
- Actividades de SPIES.
- Estimaciones de tamaño de los productos.
- Costo de adquirir externamente los productos.
- Capacidad de las herramientas proporcionadas en el entorno de ingeniería.

5.1.2. DESARROLLAR EL PLAN DEL PROYECTO

El plan del proyecto es establecido como base para controlar el proyecto

Un plan de proyecto es un documento formal, aprobado y usado para gestionar y controlar la ejecución del proyecto. El plan está basado en los requisitos del proyecto y en las estimaciones establecidas. El plan del proyecto debe considerar todas las fases del ciclo de vida propuesto por SPIES.

5.1.2.1. Establecer el presupuesto y el calendario del proyecto

Establecer el presupuesto y el calendario del proyecto

El presupuesto y el calendario del proyecto están basados en las estimaciones desarrolladas y aseguran que la asignación del presupuesto, la complejidad de las tareas y las dependencias entre éstas sean tratadas correctamente.

Los calendarios orientados-por-eventos y limitados-por-recursos han demostrado ser eficaces para tratar con los riesgos de cualquier tipo de proyecto. La identificación de los logros que deben cumplirse antes que un evento inicie, proporciona flexibilidad en el tiempo de realización, el entendimiento común de lo que se espera, una mejor visión del estado del proyecto, y un estado más exacto de las tareas del mismo.

PRODUCTOS A OBTENER:

1. Calendario del proyecto.
2. Dependencias del calendario.
3. Presupuesto del proyecto.

Actividades específicas

1. Identificar los hitos importantes

Los hitos a menudo son impuestos para asegurar la terminación de ciertos entregables. Los hitos pueden basarse en los eventos o en el calendario. Si están basados en calendario, una vez que las fechas del hito han sido acordadas debe ser muy difícil cambiarlas.

2. Identificar las suposiciones sobre el calendario

Cuando los calendarios son inicialmente desarrollados, es común hacer suposiciones sobre la duración de ciertas actividades. Estas suposiciones son hechas frecuentemente sobre elementos para los que cualquier dato de la estimación está disponible. La identificación de estas suposiciones proporciona la intuición sobre el nivel de confianza en el calendario general.

3. Identificar las dependencias entre tareas

Regularmente, las tareas para un proyecto pueden ser completadas en alguna secuencia ordenada que minimizará la duración del proyecto. Esto involucra la identificación de las tareas predecesoras y sucesoras para determinar el orden óptimo.

Dentro de los ejemplos de herramientas que pueden ayudar a determinar un orden óptimo de las actividades se incluyen los siguientes:

- Método de la Ruta Crítica (CPM)
- Técnica de Evaluación y Revisión del Programa (PERT)

4. Definir el presupuesto y calendario (Artefacto SCHE)

El establecimiento y mantenimiento del presupuesto y calendario del proyecto regularmente incluyen lo siguiente:

- La determinación del tiempo de las actividades.
- La definición de las dependencias entre las actividades (relaciones predecesora o sucesora).
- La definición de las actividades e hitos del calendario para apoyar la exactitud en la medición del progreso.
- La identificación de hitos para la generación de los productos.
- La definición de actividades con duración apropiada.
- La definición de hitos con una separación apropiada de tiempo.
- El uso de datos históricos apropiados para verificar el calendario.
- La documentación de las suposiciones y razones fundamentales del proyecto.

5. Establecer criterios de acción correctiva

Los criterios son establecidos para determinar qué constituye una desviación importante del plan del proyecto. Es necesaria una base para medir los asuntos y problemas para determinar cuándo se debe tomar una acción correctiva. Las acciones correctivas pueden requerir la replanificación, lo que puede incluir el revisar el plan original o incluir actividades de mitigación dentro del plan actual.

5.1.2.2. Identificar los riesgos del proyecto

Identificar y analizar los riesgos del proyecto

Los riesgos son identificados y analizados para apoyar a la planificación del proyecto. Esta actividad específica debe ampliarse a todos los planes que afectan al proyecto. La identificación y el análisis de los riesgos en la planificación del proyecto incluyen regularmente lo siguiente:

- Determinar el impacto, probabilidad de ocurrencia, y el marco de tiempo en el cual pueden ocurrir los problemas.
- Priorizar los riesgos.

PRODUCTOS A OBTENER:

1. Riesgos identificados.
2. Impacto de los riesgos y probabilidad de ocurrencia.
3. Prioridad de los riesgos.

Actividades específicas

1. Identificar los riesgos

La identificación de los riesgos involucra la identificación de problemas potenciales, peligros, amenazas, vulnerabilidades, y demás que podrían afectar negativamente los esfuerzos y planes del trabajo. Los riesgos deben ser identificados y descritos en una forma entendible antes de que puedan ser analizados.

2. Documentar los riesgos (Artefacto RIE)

3. Revisar los riesgos cuando sea necesario

Dentro de los ejemplos de cuándo se puede requerir la revisión de los riesgos se incluyen los siguientes:

- Cuando son identificados nuevos riesgos
- Cuando los riesgos llegan a ser problemas
- Cuando los riesgos son retirados
- Cuando las circunstancias del proyecto cambian significativamente

5.1.2.3. Planificar las necesidades del conocimiento

Planificar las necesidades de conocimientos para el desarrollo del proyecto

La entrega de conocimiento a los proyectos incluye el entrenamiento al personal del proyecto y la adquisición de conocimientos de fuentes externas. La planificación de las necesidades del conocimiento del proyecto incluyen regularmente lo siguiente:

- La formación del personal del proyecto y la adquisición de los conocimientos desde fuentes externas.
- Considerar que los requisitos de personal dependen del nivel y conocimiento disponible para dar soporte a la ejecución del proyecto.

PRODUCTOS A OBTENER:

1. Inventario del conocimiento necesario (Artefacto HAB).
2. Planes de personal y de nuevas contrataciones.

Actividades específicas

1. Identificar el conocimiento y las habilidades necesarias para ejecutar el proyecto.
2. Evaluar el conocimiento y las habilidades disponibles.
3. Seleccionar los mecanismos para proporcionar el conocimiento y las habilidades necesarias.

Algunos ejemplos de mecanismo son:

- Formación interna (de la organización y del proyecto)
- Formación externa
- Personal y nuevas contrataciones
- Adquisición externa de habilidades

4. Incorporar los mecanismos seleccionados en el plan del proyecto.

5.1.2.4. Establecer el plan del proyecto

Establecer y mantener el contenido del plan del proyecto

Es necesario un plan documentado que trate todo los elementos importantes de la planificación para alcanzar el mutuo entendimiento, compromiso y rendimiento de las personas, grupos y organizaciones que deben ejecutar o apoyar los planes. El plan generado para el proyecto define todos los aspectos del esfuerzo, unidos de una manera lógica: consideraciones del ciclo de vida del proyecto; tareas técnicas y de gestión; presupuesto y calendario; hitos; e identificación de los riesgos. Las descripciones de infraestructura incluyen relaciones de responsabilidad y relaciones para el personal del proyecto, gestión y organización de soporte.

PRODUCTOS A OBTENER

1. Plan general del proyecto (Artefacto PRY)

5.2. ESPECIFICACIÓN DE REQUISITOS

Un área de proceso de la categoría de Ingeniería de SPIES.

PROPÓSITO

El propósito de la Especificación de Requisitos es producir y analizar los requisitos del cliente, del producto y de los componentes del producto.

NOTAS INTRODUCTORIAS AL ÁREA DE PROCESO

Generalmente se acepta que los requisitos son la base para desarrollar un producto, pero también es sabido que una mala gestión de los requisitos es la principal causa de que un sistema falle.

El objetivo principal de la Especificación de Requisitos es establecer un entendimiento entre el cliente o usuario del sistema y el grupo de desarrollo con el fin de establecer cuales son los requisitos del sistema, los cuales se clasifican en operacionales, funcionales, de calidad de servicio (o QoS), paramétricos y de diseño. Es posible que se reciban requisitos de distintos sitios siempre y cuando provengan de un proveedor autorizado de requisitos, y es entonces que son incorporados al proyecto para su análisis. Una vez que los requisitos han sido documentados y aceptados por todas las partes afectadas (cliente, usuarios, grupo de desarrollo) inicia la fase de modelado la cual incluye el documentar la razón fundamental de cada requisito a lo largo de todo el ciclo de vida del desarrollo del sistema, desde su obtención hasta su implementación.

Una identificación eficaz de requisitos es fundamental para obtener un sistema correcto, es inimaginable que una pobre comprensión de los requisitos conduzca a un producto prometedor. Los requisitos son la base para el diseño del sistema. La especificación de los requisitos incluye las siguientes actividades:

- Educción, análisis, validación y comunicación de las necesidades, las expectativas y las restricciones del sistema a construir.

- Recogida y coordinación de las necesidades de los participantes en el proyecto.
- Desarrollo de los requisitos.
- Establecimiento de los requisitos iniciales del producto y de los componentes del producto consistentes con la especificación del mismo.

Los análisis se usan para comprender, definir y seleccionar los requisitos a todos los niveles a partir de distintas alternativas. Estos análisis incluyen:

- Análisis de necesidades y de requisitos para cada fase del ciclo de vida de SPIES, incluyendo el entorno operativo y factores tales como seguridad y protección
- Desarrollo de un concepto operativo
- Definición de la funcionalidad requerida

La definición de funcionalidad (o análisis funcional) no es lo mismo que el análisis estructurado en el desarrollo de software y no supone un diseño de software orientado a la funcionalidad. En el diseño de software orientado a objetos, por ejemplo, la funcionalidad se relaciona con la definición de los llamados "servicios" o "métodos". En SPIES, la definición de funciones, sus agrupaciones lógicas y su asociación con los requisitos se refiere como "arquitectura funcional".

Los análisis en SPIES ocurren incrementalmente y en capas sucesivas de más detalle de una arquitectura del producto hasta que se alcanza el suficiente detalle para permitir el diseño detallado y las pruebas del producto a seguir. Como resultado de este análisis de requisitos y del concepto operativo (incluyendo la funcionalidad), la fabricación o el concepto de producción del sistema empotrado produce más requisitos derivados, incluyendo consideraciones de:

- Restricciones de varios tipos.
- Limitaciones tecnológicas.
- Costos y parámetros de costo.
- Restricciones de tiempo y parámetros del calendario Riesgos.
- Consideraciones sobre problemas implícitos, pero no declarados explícitamente por la especificación o por el usuario final.
- Factores introducidos por consideraciones únicas del desarrollador, por regulaciones y por leyes.

Así, una jerarquía de entidades lógicas (funciones y subfunciones, clases y subclases de objetos) se establece a través de cada incremento con la evolución del concepto operativo. Los requisitos son refinados, se derivan y se asignan a estas entidades lógicas. Los requisitos y las entidades lógicas se asignan a los productos, a los componentes del producto, al personal o a los procesos asociados.

ÁREAS DE SPIES RELACIONADAS

Consultar el área de Integración del Producto para mayor información sobre los requisitos de la interfaz y la gestión de la misma.
Consultar el área de Validación del Producto para mayor información sobre cómo será validado el producto construido frente a las necesidades del cliente.
Consultar el área de Gestión de la Configuración para mayor información sobre el control y gestión de los productos de trabajo.

RESUMEN DE OBJETIVOS

1. Desarrollar los requisitos del producto
    1.1. Obtener las necesidades.
    1.2. Desarrollar los requisitos.
    1.3. Establecer los requisitos del producto y de componentes del producto.
    1.4. Identificar los requisitos de interfaz.
2. Analizar y validar los requisitos
    2.1. Establecer los conceptos operativos y los escenarios.
    2.2. Establecer una definición de funcionalidad requerida.
    2.3. Analizar y validar los requisitos.

ACTIVIDADES POR OBJETIVO

5.2.1. DESARROLLAR LOS REQUISITOS DEL CLIENTE

Las necesidades, expectativas, restricciones e interfaces de las partes interesadas son recogidas y traducidas a requisitos del producto

Las necesidades del producto (provenientes del cliente o especificación escrita) son la base para determinar los requisitos del mismo. Las necesidades, las expectativas, las restricciones, las interfaces, los conceptos operativos y los conceptos de producto son analizados, unificados, refinados y elaborados para la traducción de un conjunto de requisitos.

SPIES utiliza un proceso iterativo durante toda la vida del proyecto para identificar y comprender claramente las necesidades del producto. Para facilitar la interacción requerida, normalmente se involucra a un sustituto del usuario final para representar sus necesidades y para ayudar en la resolución de conflictos. Las restricciones ambientales, legales y otras deberían considerarse al crear y resolver el conjunto de requisitos del producto.

5.2.1.1. Obtener las necesidades

Obtener las necesidades, las expectativas, las restricciones, y las interfaces para todas las fases del ciclo de vida del producto

La obtención va más allá de la recogida de requisitos mediante la identificación de requisitos adicionales no proporcionados por la especificación del producto o el cliente.

Dentro de los ejemplos de técnicas para obtener las necesidades se incluyen los siguientes:

- Revisiones intermedias del proyecto
- Cuestionarios, entrevistas y escenarios operativos obtenidos con los usuarios finales

- Prototipos y modelos
- Brainstorm
- Estudios de mercado
- Pruebas beta
- Extracción de fuentes tales como documentos, estándares o especificaciones
- Observación de productos, entornos y patrones de flujo existentes
- Casos de uso
- Ingeniería inversa

SPIES utiliza la técnica de los casos de uso para determinar lo que debe hacer el producto y cómo se comportará.

Dentro de los ejemplos de fuentes de requisitos que podrían no ser identificadas al inicio se incluyen los siguientes:

- Políticas de negocio
- Estándares
- Requisitos ambientales de negocio (p.e. laboratorios, pruebas y otras instalaciones)
- Tecnología
- Productos o componentes del producto heredados

PRODUCTOS A OBTENER:

1. Introducción funcional.
2. Diagramas de casos de uso.

Actividades específicas

1. Preparar el entorno de trabajo para la Introducción Funcional (Artefacto DCUF)

Los Diagramas de Casos de Uso (DCU) muestran las funciones del sistema y las entidades que se encuentran fuera del sistema (actores). Los DCU permitirán obtener la especificación de los requisitos para el sistema y mostrar las interacciones entre el sistema y los factores externos. SPIES utiliza Rhapsody© de Telelogic para realizar la especificación de requisitos y el diseño del sistema, por lo que es necesario seguir paso a paso las actividades de esta área de proceso.

2. Crear un nuevo DCU en el paquete de Análisis

En el explorador de Rhapsody, es necesario expandir la categoría de Paquetes. Se deberá escoger el paquete de Análisis y utilizar el menú contextual que surge de utilizar el botón derecho del mouse para seleccionar *Add New* > *Use Case Diagram*. Este nuevo diagrama deberá recibir el nombre de Introducción Funcional.

Rhapsody agregará automáticamente al explorador la categoría de Diagramas de Caso de Uso y el nombre del diagrama y abrirá una nueva área de dibujo. Antes de obtener el DCU de la Introducción Funcional es necesario identificar los requisitos del sistema incluyendo los actores, los casos de uso más importantes, y las relaciones entre ellos.

3. Dibujar la Caja Límite y los Actores

La Caja Límite delimita el sistema que se está diseñando de los actores externos. Los casos de uso estarán dentro de la Caja Límite; los actores estarán fuera de la Caja Límite.

Figura 5.1. Componentes utilizados para dibujar la Caja Límite y los Actores

i. Con el botón izquierdo del mouse, presionar el icono *Caja Limite* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presione *Actor* que está en la barra de dibujo.

Dado que la generación de código utiliza los nombres especificados al crear un actor, se recomienda no incluir espacios en los nombres de los mismos.

Es posible utilizar las herramientas del *Botón de Alineación* para alinear los elementos que se seleccionen del diagrama. Por ejemplo, se pueden utilizar el botón de alineación para alinear dos elementos seleccionados con respecto al límite inferior del área de dibujo, o *Botón Escala*, para hacerlos del mismo tamaño.

iii. Con el botón izquierdo del mouse, presionar en el área de dibujo para insertar el actor y darle un nombre relacionado con su rol. Es importante recordar que los actores deben dibujarse fuera de la Caja Límite. Será necesario repetir esta actividad de acuerdo al número de actores que se desee introducir.

4. Dibujar los casos de uso

Un caso de uso representa una función particular del sistema.

Figura 5.2. Componentes utilizados para dibujar los casos de uso

i. Con el botón izquierdo del mouse, presionar el icono *Caso de Uso* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar en el área de dibujo para insertar el caso de uso dentro de la Caja Límite y darle un nombre relacionado con la función a la que se relaciona. Será necesario repetir esta actividad de acuerdo al número de casos de uso que se desee introducir.

> Es posible utilizar espacios para los nombres de los casos de uso. A diferencia de los actores, los casos de uso no corresponden con el código que será generado.
>
> Al crear un caso de uso, en el explorador de Rhapsody se agregará automáticamente la categoría de *Uses Cases* en el paquete de Análisis.

5. Definir las propiedades de los casos de uso

i. Si es necesario, expandir en el explorador el paquete de Análisis y la categoría *Use Cases*.

ii. Presionar dos veces el botón izquierdo del mouse sobre el caso de uso que se quiera definir o utilizar el botón derecho del mouse sobre el nombre del caso de uso que aparece en la categoría expandida y seleccionar Features.

iii. Documentar el caso de uso utilizando la pestaña Description de la ventana de *Features*.

iv. Será necesario repetir las tres actividades anteriores para documentar cada uno de los casos de uso.

> Es posible también utilizar el icono *Más Información* de la caja de Description dentro de *Features* para expandir el editor de texto. Las opciones de *Features* quedarán registradas cuando se presionen los botones *OK* o *Apply*.
>
> Con la ventana de *Features* abierta, es posible documentar cada uno de los casos de uso sin cerrarla; solo se debe seleccionar un caso de uso distinto. Al terminar con todos los casos de uso es necesario aceptar los cambios.

6. Asociar los actores con los casos de uso

Es necesario mostrar las *asociaciones* entre los actores y los casos de uso relevantes utilizando líneas de asociación. Una asociación representa una conexión entre los objetos y los usuarios. Para dibujar una asociación es necesario:

i. Con el botón izquierdo del mouse, presionar el icono *Asociación* que está en la barra de dibujo. Una vez que se mueva el cursor por el área de dibujo éste se convertirá en una cruz que indica que se ha activado el proceso de trazado y cambiará a un círculo con cruz cuando se permita trazar la asociación.

ii. Mover el cursor hacia el borde de un actor (o caso de uso) y presionar el botón izquierdo del mouse para indicar el trazado de la asociación. Al alcanzar el borde del actor (o caso de uso) con el que se desea conectar, será necesario presionar de nuevo el botón izquierdo del mouse para terminar con la asociación. Será necesario repetir esta actividad de acuerdo al número de asociaciones que se desee introducir.

> Al crear una asociación, en el explorador de Rhapsody se agregará automáticamente la categoría de Actores en el paquete de Análisis que muestra las asociaciones entre los actores y los casos de uso.

7. Dibujar las generalizaciones

Una *generalización* es una relación entre un elemento general y un elemento más específico. El elemento más específico hereda las propiedades del elemento general y es substituible por el elemento general. Una generalización permite la derivación de un caso de uso en otro.

i. Con el botón izquierdo del mouse, presionar el icono *Generalización* que está en la barra de dibujo. Es necesario desplazar el cursor hacia el caso de uso específico, presionar el botón izquierdo en uno de sus bordes, y mostrar la dirección del trazado de la generalización. Cuando se alcance el borde del caso de uso elemental es necesario presionar de nuevo el botón izquierdo del mouse. Será necesario repetir esta actividad de acuerdo al número de generalizaciones que se desee introducir.

8. Documentar los modelos y diagramas

Una de las principales propiedades de SPIES consiste en documentar los modelos y diagramas obtenidos. Esto facilita la reutilización de componentes en proyectos futuros. Rhapsody soporta los siguientes tipos de observaciones:

Figura 5.3. Componentes utilizados para documentar los modelos y diagramas

- Notas: contienen información que puede ser de utilidad a los lectores
- Limitaciones: son condiciones o restricciones expresadas en texto
- Comentarios: extiende la explicación sobre algún caso de uso en particular
- Requisitos: son anotaciones textuales que describen la intención de un elemento
- Ancla: permite adjuntar una limitación, comentario, requisito, o nota a uno o más elementos.

i. Con el botón izquierdo del mouse, presionar el icono de acuerdo al tipo de observación que se desee agregar (Notas, Limitaciones, Comentarios, Requisitos, o Anclas).

ii. Con el botón izquierdo del mouse, presionar en el área de dibujo (fuera de la Caja Límite) para agregar la observación escogida.

iii. Introducir la descripción de la observación por cada observación en el diagrama.

5.2.1.2. Desarrollar los requisitos

Transformar las necesidades, expectativas, restricciones y las interfaces de los participantes en el proyecto en requisitos del sistema

El modelado de los requisitos habilitará la trazabilidad de los requisitos sin utilizar una herramienta especializada para la Gestión de los Requisitos. La trazabilidad de los requisitos es la habilidad para describir y seguir la vida de un requisito, en ambas direcciones (hacia adelante y hacia atrás). La trazabilidad apoya la validación y verificación de los requisitos, previene la introducción de características no especificadas, y proporciona la visibilidad para los requisitos derivados que necesitan ser especificados y probados.

PRODUCTOS A OBTENER:

1. Diagramas de requisitos

Actividades específicas

1. Agregar los requisitos al modelo

Los requisitos pueden ser representados como elementos del requisito. Para cada requisito del sistema es necesario:

i. En el explorador de Rhapsody, seleccionar el paquete de Requisitos.

ii. Seleccionar *Add New* > *Requirement* del menú contextual del paquete.

iii. Nombrar el requisito.

iv. Documentar el requisito presionando dos veces el botón izquierdo del mouse o utilizando el menú contextual del requisito.

Es necesario utilizar una notación sencilla que permita diferenciar entre los requisitos según crezca el análisis y descripción de los mismos. SPIES recomienda utilizar una notación simple que comienza con el vocablo "Req." (indicando así que se trata de un requisito) para continuar con una numeración que distingue entre componente y requisito. Es decir, el Req.1.1 indicaría que se trata del requisito 1 (número a la derecha del punto decimal) del componente 1 (número a la izquierda del punto decimal).

2. Agregar los elementos del requisito

Es posible agregar elementos del requisito que muestren cómo se relacionan los requisitos con los casos de uso. En diagramas de alto nivel es posible agregar requisitos directamente sobre los casos de uso, si el diseño lo requiere.

i. Expandir la categoría de Requisitos del paquete de Requisitos.

ii. Con el botón izquierdo del mouse, seleccionar el requisito que se desee insertar como elemento y arrastrarlo hacia el DCU.

Rhapsody permite establecer qué tipo de información será desplegada en los elementos del modelo y el formato gráfico para su visualización mediante el cuadro de dialogo de *Display Options*. Para establecer las opciones es necesario que, con el botón derecho del mouse, se utilice el menú contextual del requisito para ajustar sus propiedades.

Display Options permite configurar dos tipos de propiedades: *Show* que especifica la información de visualización del requisito (si se desea visualizar solo el nombre del requisito omitiendo información adicional como la descripción, por ejemplo, se deberá seleccionar la opción *Name*) y *Form* que especifica la forma gráfica en que será visualizado el elemento requisito.

Los cambios de *Display Options* quedarán registrados cuando se presione el botón *OK*.

3. Dibujar las dependencias entre los requisitos

Una *dependencia* es una relación directa en la cual la función de un elemento requiere la presencia de y puede cambiar otro elemento. Las dependencias sirven para mostrar las relaciones entre los requisitos, y entre los requisitos y los elementos del modelo.

i. Con el botón izquierdo del mouse, presionar el icono *Dependencia* que está en la barra de dibujo. Es necesario desplazar el cursor hacia el requisito que será el origen de la relación, presionar el botón izquierdo en uno de sus bordes, y dibujar la línea hacia el otro requisito o caso de uso que forma parte de la dependencia. Este paso se deberá repetir para todas las dependencias que se deseen agregar.

En el explorador de Rhapsody se puede expandir la categoría de Requisitos para visualizar las relaciones de dependencia del modelo.

4. Definir el estereotipo de las dependencias

El estereotipo es la forma de especificar las formas en que se relacionan los requisitos con otros requisitos y elementos del modelo. Un estereotipo es un elemento del modelado que extiende la semántica del metamodelo de UML mediante el uso de entidades.

i. Seleccionar la dependencia a la que se desea agregar el estereotipo y presionar dos veces el botón izquierdo del mouse o el botón derecho para el menú contextual.

ii. En la pantalla *General de Features*, seleccionar el estereotipo de la lista desplegable que más se adecue al análisis. Los cambios de *Features* quedarán registrados cuando se presione el botón *OK*.

5. Definir los diagramas de requisitos (Artefacto DREQ)

Los diagramas de requisitos son esquemas de análisis a un nivel de mayor detalle. Estos diagramas están compuestos exclusivamente por requisitos y no muestran interacciones con los casos de uso.

i. Con el botón izquierdo del mouse, presionar sobre el paquete de Requisitos y seleccionar *Add New > Use Case Diagram*.

ii. Nombrar el nuevo diagrama y presionar *OK* para crear la nueva área de modelado.

iii. En el explorador de Rhapsody, expandir la categoría de Requisitos en el paquete de Requisitos.

iv. Seleccionar los requisitos que serán parte del diagrama de Requisitos y arrastrarlos individualmente al área de modelado.

v. Con el botón derecho del mouse, presionar sobre cada requisito para escoger del menú contextual *Display Options*.

vi. Del grupo de opciones *Show*, seleccionar la opción *Name*. Los cambios de *Display Options* quedarán registrados cuando se presione el botón *OK*.

vii. Dibujar dependencias para indicar la relación entre los requisitos del diagrama y escoger el estereotipo para cada una de ellas, de acuerdo al análisis del sistema.

Rhapsody agrega automáticamente al explorador las relaciones de dependencia.

5.2.1.3. Establecer los requisitos del producto y de componentes del producto

Los requisitos del sistema son refinados y elaborados para desarrollar los requisitos del producto y componentes del producto

Los requisitos del sistema se analizan conjuntamente con el desarrollo del concepto operativo para derivar conjuntos de requisitos más detallados y precisos llamados "requisitos del producto y de componentes del producto". Estos requisitos tratan las necesidades asociadas con cada fase del ciclo de vida del producto. Los requerimientos derivados surgen de restricciones, considerando los problemas implícitos pero declarados explícitamente en los requisitos base (fiabilidad, seguridad, QoS, rapidez, y demás). Los requisitos son revisados con el diagrama de requisitos de bajo nivel, con la estrategia de desarrollo (Artefacto STR), y el concepto de producto preferido se refina. Los requisitos de un sistema empotrado pueden ser:

- Requisitos operacionales que son aquellos que especifican cómo colabora el sistema con otros elementos (actores) en su entorno.
- Requisitos funcionales que tienen que ver con el comportamiento del sistema. Si un requisito funcional fuera un verbo, entonces un requisito QoS sería un adverbio dado que especifica el "cuánto".
- Requisitos paramétricos que son tanto "no operacionales" como "no funcionales".
- Requisitos de diseño que tienen que ver con las características del diseño mismo pero no del sistema.

PRODUCTOS A OBTENER:

1. Requisitos derivados.
2. Requisitos del producto.
3. Requisitos de componentes del producto.
4. Formato de asignación de requisitos.
5. Restricciones de diseño.

Actividades específicas

1. Desarrollar los requisitos en los términos técnicos necesarios para el diseño del sistema y sus componentes

Desarrollar los requisitos de la arquitectura que cubran las necesidades de calidad y rendimiento del producto y que son necesarios para la arquitectura del sistema. La estrategia de desarrollo deberá ser refinada con una visión más clara de funcionalidad.

2. Derivar los requisitos resultantes de las decisiones de diseño

La selección de una tecnología trae consigo requisitos adicionales. Por ejemplo, el uso de la electrónica agrega requisitos específicos.

Los requisitos de arquitectura expresan los puntos de calidad y de rendimiento que son críticos para el éxito del producto.

3. Asignar los requisitos para cada componente del producto (Artefacto ASIREQ)

Algunas veces un requisito de nivel más alto especifica el rendimiento que se satisface por múltiples componentes del sistema. A menudo se realiza una asignación provisional de un requisito de nivel más alto a componentes del producto, pero se corrige en la fase de Diseño del Producto para considerar otros aspectos funcionales.

i. Asignar los requisitos a las funciones.

ii. Asignar los requisitos a los componentes del producto.

iii. Asignar las restricciones de diseño a los componentes del producto.

5.2.1.4. Identificar los requisitos de interfaz

Identificar los requisitos relacionados con la interfaz lógica del sistema

PRODUCTOS A OBTENER:

1. Requisitos de la interfaz (Artefacto ASIREQ)

Actividades específicas

1. Identificar las interfaces tanto externas como internas al sistema

A medida que el área de proceso de Diseño del Producto avanza, la arquitectura del producto será modificada por el área de proceso de Desarrollo del Producto, creando posiblemente nuevas interfaces entre los componentes del sistema y los componentes externos al mismo.

2. Desarrollar los requisitos para las interfaces identificadas

Los requisitos de las interfaces se definen en términos tales como las características de los datos para el software, y las características eléctricas y mecánicas para el hardware.

5.2.2. ANALIZAR Y VALIDAR LOS REQUISITOS

Los requisitos son analizados y validados, y una definición de la funcionalidad requerida es desarrollada

Los análisis se ejecutan para determinar qué impacto tendrá el entorno operativo previsto sobre la capacidad para satisfacer las necesidades, expectativas, restricciones y las interfaces del sistema. Los análisis examinan los requisitos desde diferentes perspectivas (viabilidad, costo y riesgo) y pueden utilizar diferentes abstracciones (funcional, flujo de datos, entidad-relación, y diagramas de estados).

5.2.2.1. Establecer los conceptos operativos y los escenarios

Establecer y mantener los conceptos operativos y los escenarios asociados

Un escenario normalmente es una secuencia de eventos que podrían ocurrir en el uso del producto. Como contraste, un concepto operativo para un producto depende generalmente tanto de la solución de diseño como del escenario. Los escenarios pueden incluir secuencias operativas las cuales son una expresión de los requisitos del sistema más que conceptos operativos.

PRODUCTOS A OBTENER:

1. Concepto operativo (Artefacto ASIREQ).
2. Casos de uso refinados (Artefacto DCUF).
3. Escenarios.
4. Nuevos requisitos.

Actividades específicas

1. **Desarrollar los conceptos operativos y los escenarios que incluyan funcionalidad, rendimiento y soporte según sea apropiado.**
2. **Definir el entorno en el cual funcionarán el producto o los componentes del producto, incluyendo límites y restricciones.**
3. **Revisar los conceptos operativos y los escenarios para refinar y descubrir los requisitos.**

El concepto operativo y el desarrollo del escenario es un proceso iterativo que será mejorado en el área de proceso de Diseño del Producto.

5.2.2.2. Establecer una definición de funcionalidad requerida

Establecer y mantener una definición de la funcionalidad requerida por el sistema

La definición de la funcionalidad, también referida como "análisis funcional", es la descripción de lo que se pretende que haga el sistema. La definición de la funcionalidad puede incluir acciones, secuencia, entradas, salidas u otra información que comunique la manera en la cual el sistema será usado.

La arquitectura funcional será definida formalmente en el área de proceso de Diseño del Producto.

PRODUCTOS A OBTENER:

1. Funcionalidad requerida (Artefacto ASIREQ).

Actividades específicas

1. **Analizar y establecer la funcionalidad requerida por los usuarios finales.**
2. **Dividir los requisitos en grupos, en base a los criterios establecidos (funcional, no funcional, seguridad, no importante).**
3. **Asignar los requisitos funcionales a los componentes o funciones del sistema.**

5.2.2.3. Analizar y validar los requisitos

Analizar y validar los requisitos para asegurarse que son necesarios y suficientes, y asegurar que el sistema resultante funcionará según lo previsto por el usuario

El análisis de los requisitos ayuda a contestar a preguntas tales como si todos los requisitos son necesarios, si falta alguno, si son consistentes unos con otros, y si pueden implementarse y verificarse. A medida que los requisitos son definidos, su relación con requisitos de más alto nivel y la funcionalidad definida de más alto nivel deben comprenderse.

PRODUCTOS A OBTENER:

1. Informes de defectos en los requisitos y cambios propuestos para resolver los defectos (Artefacto ANAREQ)
2. Simulación

Actividades específicas

1. **Analizar los requisitos (Artefacto ASIREQ) para determinar si éstos satisfacen los objetivos de los requisitos de nivel más alto.**
2. **Analizar los requisitos para asegurarse de que son completos, factibles, realizables y verificables**
3. **Documentar las modificaciones para resolver los defectos encontrados.**
4. **Explorar la adecuación y la completitud de los requisitos desarrollando la representación del sistema a través de simulaciones, y obteniendo retroalimentación sobre ellos ya sea por el analista o los usuarios finales.**

Para más información sobre la simulación con escenarios consulte el área de proceso de Diseño del Producto.

5.3. DISEÑO DEL PRODUCTO

Un área de proceso de la categoría de Ingeniería de SPIES

PROPÓSITO

El propósito del Diseño del Producto consiste en simplificar la realización de las tareas propias del desarrollo del producto mediante la reducción del esfuerzo.

NOTAS INTRODUCTORIAS AL ÁREA DE PROCESO

El diseño de un producto o sistema regularmente es considerado una actividad meramente creativa que es difícil de dividir en pasos pequeños, fáciles de realizar. Así como los requisitos de un sistema forman la base para las decisiones sobre su arquitectura, tales decisiones limitan a su vez la arquitectura del software.

Como ya se menciono, en el diseño de los sistemas empotrados se dan cita dos elementos conceptuales. Por una parte, está la arquitectura hardware construida en torno a un sistema basado en microprocesador, y por otra la arquitectura software que involucra los sistemas operativos, los lenguajes de programación, compiladores, herramientas de modelado, simulación y evaluación. La integración de la arquitectura hardware y software proporciona la arquitectura global del sistema empotrado.

En algunos casos, una unidad diferente de la organización diseña la arquitectura del sistema. Así pues, la arquitectura está más o menos establecida –por ejemplo, cuando la arquitectura de hardware fue diseñada primero o ya era conocida. Esto lleva a las arquitecturas subóptimas (de software). Dado que el software es considerado más flexible y tiene un plazo de ejecución más corto, los proyectos lo utilizan para arreglar los defectos de la arquitectura de hardware, tal y como se mencionó antes. El proceso de diseño no siempre considera de forma explícita los requisitos de rendimiento. En la mayoría de los casos donde el rendimiento es un problema, los sistemas son diseñados para ser lo más rápidos posibles. Es decir, no se establece "qué tan rápidos" serán hasta que la implementación está disponible. Aquellos proyectos que consideran los requisitos de rendimiento durante el diseño lo hacen en base al presupuesto. Por ejemplo, con frecuencia se divide una limitación de alto nivel en tiempo real entre varios componentes de nivel inferior. Esta división, sin embargo, a menudo está basada en la experiencia de los desarrolladores en lugar de cálculos bien fundamentados. Los proyectos también suelen utilizar esta técnica para otros requisitos no funcionales como la energía y la memoria.

La arquitectura del software a menudo se refleja en la arquitectura del hardware, lo que facilita el determinar el impacto de los cambios en el hardware. La mayoría de los casos que implican sistemas complejos emplean un patrón de la arquitectura en capas. Estas capas simplifican la labor de ocuparse de la complejidad creciente de los sistemas empotrados.

ÁREAS DE SPIES RELACIONADAS

Consultar el área de Especificación de Requisitos para mayor información sobre la definición y desarrollo de los requisitos que definen el producto y componentes del producto. Estos requisitos (de producto y componentes) sirven como base para el diseño de la arquitectura del software y del hardware.

Resumen de Objetivos:

1. Diseñar la arquitectura del sistema
    1.1. Diseñar los diagramas de estructura
    1.2. Asignar las funciones entre los subsistemas
2. Crear la arquitectura de los subsistemas
    2.1. Diseñar el diagrama del modelo de objetos.
    2.2. Generar el código y compilar el modelo.
3. Crear los diagramas de secuencia
    3.1. Diseñar los diagramas de secuencia.
    3.2. Validar los diagramas de secuencia.
4. Crear los diagramas de actividades
    4.1. Diseñar los diagramas de actividades.
    4.2. Validar los diagramas de actividades.
5. Crear los diagramas de máquinas de estados
    5.1. Diseñar los diagramas de máquinas de estados.
    5.2. Validar los diagramas de máquinas de estados.

ACTIVIDADES POR OBJETIVO

5.3.1. DISEÑAR LA ARQUITECTURA DEL SISTEMA

Diseñar los diagramas que definen la estructura del sistema e identifican las piezas organizacionales a larga escala del sistema

Los diagramas de estructura muestran el flujo de la información entre los componentes del sistema y la definición de la interfaz a través de los puertos. Es sistemas grandes, a menudo los componentes son descompuestos en funciones o módulos subsistema.

5.3.1.1. Diseñar los diagramas de estructura

Diseñar los diagramas de estructura que muestren la partición del sistema especificado por los requisitos y su jerarquía

En un diseño de alto nivel, un diagrama de estructura identifica los componentes (objetos) del sistema y describe el flujo de datos entre los componentes desde una perspectiva de caja-negra.

PRODUCTOS A OBTENER:

1. Diagramas de estructura (Artefacto DEST).

Actividades específicas

1. Establecer la configuración del diagrama de estructura

i. En el explorador de Rhapsody, expandir la categoría de Paquetes y seleccionar el paquete de Arquitectura.

ii. Seleccionar *Add New* > *Structure Diagram* del menú contextual del paquete.

iii. Nombrar el diagrama.

Rhapsody crea automáticamente la categoría de Diagramas de Estructura en el explorador y agrega el nombre al diagrama creado. Además, habilita la nueva área de dibujo para el diagrama de estructura.

Figura 5.4. Componentes utilizados para dibujar los diagramas de estructura

2. 2. Dibujar los objetos

Un *objeto* es una entidad con límites bien definidos y una identidad que encapsula el estado y el comportamiento. Un estado es representado por los atributos y las relaciones, mientras que un comportamiento es representado por las operaciones, métodos y máquinas de estado. Un objeto es una instancia de una clase. En lenguajes orientados-a-objetos como C++, una clase es una plantilla para la creación de instancias (objetos) que comparten los mismos atributos, operaciones, y métodos.

i. Con el botón izquierdo del mouse, presionar el icono *Objeto* que está en la barra de dibujo.

ii. Arrastrar el objeto al área de dibujo y renombrarlo de acuerdo con el diseño.

iii. Será necesario repetir las tareas anteriores de acuerdo a la cantidad de objetos que se desee introducir al diagrama.

3. Definir objetos como subsistemas

Es necesario identificar aquellos objetos que serán subsistemas y que serán descompuestos más adelante. Es importante recordar que SPIES inculca la tarea de dividir la complejidad del sistema mediante la creación de subsistemas que serán resueltos de forma individual para al final integrar la solución completa.

i. Seleccionar el objeto al que se desea definir como subsistema del modelo y presionar dos veces el botón izquierdo del mouse o el botón derecho para el menú contextual.

ii. En la pantalla *General de Features*, en la pestaña de estereotipo, seleccionar de la lista desplegable la propiedad de *Subsystem in PredefinedTypesCpp*. Los cambios de *Features* quedarán registrados cuando se presione el botón *OK*.

4. Modificar la visualización de los subsistemas

Un objeto que ha sido definido como un subsistema debe de diferenciarse del resto de los objetos de tal forma que se logre alcanzar un modelo específico a partir de un modelo general.

i. Con el botón derecho del mouse, presionar sobre el objeto que se haya definido como subsistema y seleccionar del menú contextual la opción de *Display Options*.

ii. En el grupo de *Display Name* seleccionar la opción de *Label*.

iii. Limpiar la opción de *Show Stereotype Label* que por default viene marcada.

iv. En el grupo *Image View* seleccionar la opción *Enable Image View* y escoger la opción *Use Associated Image*.

v. Activar el botón *Advanced* y seleccionar la opción *Structured*.

vi. Presionar *OK* para guardar todos los cambios realizados a *Display Options*.

> Los objetos que han sido definidos como subsistemas son dibujados con sus etiquetas subrayadas. Este es el estilo por default para la visualización de una etiqueta. Además, los subsistemas mostrarán la imagen asociada que viene precargada en Rhapsody (esta imagen puede modificarse en la configuración de *Advanced*). El número en la esquina superior izquierda muestra la multiplicidad para el objeto.

5. Establecer estereotipos para el resto de los objetos

Es posible definir las características de un objeto, incluyendo el tipo y estereotipo. El tipo especifica la clase de la cual el objeto es una instancia; es decir, proporciona una única instancia para cada objeto.

i. Para los actores, con el botón derecho del mouse presionar para habilitar la ventana *Features*.

ii. En la pestaña General, en la opción *Stereotype* seleccionar <<*New*>> e introducir Actor en el nombre. Después de aceptar los cambios, Actor aparece en la lista como un estereotipo más.

> El paso ii será necesario siempre y cuando el modelo se esté creando desde cero puesto que no existen estereotipos creados para el paquete de Arquitectura. Para versiones posteriores que sean almacenadas y leídas constantemente, el estereotipo Actor ya aparecerá en el paquete Arquitectura y solamente tendrá que seleccionarlo de la lista desplegable de la opción *Stereotype*.

iii. En la misma pantalla General, en la opción *Type* seleccionar de la lista el nombre del actor que se está asociando en el diseño.

iv. En el grupo *Image View* seleccionar la opción *Enable Image View* y escoger la opción *Use Associated Image*.

v. Activar el botón *Advanced* y seleccionar la opción *Structured*.

vi. Presionar *Apply* para guardar todos los cambios realizados a los objetos.

> Al aplicar los cambios, Rhapsody desplegará un mensaje que indica que al definir un tipo específico para el objeto se perderán las propiedades actuales de dicho objeto. Esto es natural puesto que Rhapsody previene la modificación arbitraria de tipos por lo que solamente deberá indicarle que continué presionando el botón *Yes*.

vii. Con el botón derecho del mouse, presionar sobre un actor y seleccionar la opción *Display Options* del menú contextual.

viii. En el grupo de *Display Name* seleccionar la opción de *Label*.

ix. Limpiar la opción de *Show Stereotype* que por default viene marcada.

x. Presionar *OK* para guardar todos los cambios realizados a *Display Options*.

xi. Será necesario repetir todos los pasos anteriores para todos los actores que se hayan introducido al modelo.

6. Dibujar los puertos

Un *puerto* es una interacción distinta entre una clase o un objeto y su entorno. Los puertos permiten capturar la arquitectura del sistema mediante la especificación de interfaces entre los componentes del mismo, los cuales definen las relaciones entre los subsistemas.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Puerto* que está en la barra de dibujo.

ii. Es necesario desplazar el cursor hacia el objeto específico, presionar el botón izquierdo en uno de sus bordes en el cual se desee relacionar el objeto con el puerto.

iii. Cada que se introduce un puerto, es necesario colocar su nombre en la parte resaltada. Los nombres de los puertos deben representar la relación que existe entre los objetos que relaciona. Regularmente se sigue el patrón de funcionalidad (por ejemplo, *data_req* para indicar una petición de datos) o el patrón de objetos relacionados (por ejemplo, *mm_network* para indicar la relación entre el objeto *mm* y el objeto *network*).

Es necesario introducir todos los puertos en todos los objetos que se intente relacionar. No existen reglas sobre la cantidad de puertos que puede tener un objeto. Rhapsody agregará automáticamente en el explorador (bajo sus respectivos objetos) aquellos puertos que han sido creados.

7. Especificar los atributos de los puertos

Algunos puertos pueden ser definidos como puertos de comportamiento, los cuales indican que los mensajes enviados son retransmitidos a la clase propietaria. Con puertos que no son de comportamiento, los mensajes son enviados a una de las partes internas de la clase.

i. Con el botón derecho del mouse, presionar sobre el puerto y seleccionar *Features*.

ii. En la pestaña de General, en el grupo *Attributes*, seleccionar la opción *Behavior*.

iii. Es necesario presionar *OK* para guardar todos los cambios realizados al puerto.

8. Dibujar los flujos

Los flujos especifican el intercambio de información entre los elementos del sistema. Estos permiten describir el flujo de los datos dentro de un sistema en una etapa muy temprana (diseño de bajo nivel), antes de confiar en un diseño específico.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Flujo* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el puerto origen y presionar de nuevo el botón sobre el puerto destino.

iii. Es necesario repetir estos pasos para todos los flujos que se desee introducir.

La información puede fluir desde un elemento a otro o entre elementos en cualquier dirección. Así, es posible que el diseño exija cambiar la dirección de un flujo de datos o establecer un flujo bidireccional. Para definir flujos bidireccionales, con el botón derecho del mouse presionar sobre el flujo y seleccionar la opción *Features*. En la pestaña de General deberá seleccionar Bidirectional de la lista de *Direction*. Es necesario presionar OK para guardar los cambios sobre los flujos.

9. Especificar los ítems de los flujos

Una vez que se determina cómo ocurre la comunicación a través de los flujos, es necesario especificar la información que se transmite en el flujo utilizando los ítems. Un *ítem de flujo* puede representar cualquier dato, instanciación de dato, o eventos.

i. Con el botón derecho del mouse, presionar sobre el flujo y seleccionar *Features*.

ii. En la pestaña de *Flowitems* presionar la opción <*Add*>.

iii. En el menú de *Flowitems*, en la pestaña de General deberá dar un nombre para el ítem. Este ítem de flujo representa la información, datos, peticiones, etc. que el usuario envía al sistema.

iv. Es necesario presionar *OK* para guardar todos los cambios realizados al flujo.

v. Es necesario repetir estos pasos para todos los ítems de flujo que se desee introducir.

> Rhapsody maneja tres formas de trazo que pueden utilizarse para dibujar líneas y flechas: recta, curveada y rectilínea. Regularmente el cambio en la forma de los flujos se debe a aspectos de visualización en el modelo; para cambiar el estilo del trazo seleccione el flujo y presione el botón derecho del mouse, escoja *Line Shape* y tome la mejor opción que se adecue a la distribución de sus elementos en el modelo.

10. Especificar los contratos de los puertos

La especificación de los contratos es una actividad del área de proceso de Gestión de Contratos.

Consultar el área de Gestión de Contratos para mayor información sobre la especificación de contratos en los puertos para un diagrama de diseño.

5.3.1.2. Asignar las funciones entre los subsistemas

Dividir las operaciones del sistema en subsistemas funcionales y asignar las actividades entre estos

Una vez que se ha logrado capturar el diseño de la arquitectura del sistema en diagramas de objetos, es necesario organizar el paquete de subsistemas (*SubsystemsPkg*).

Los paquetes permiten la división del sistema en dominios funcionales o subsistemas, los cuales consisten de objetos, tipos de objetos, funciones, variables, y otros artefactos lógicos. Estos pueden ser organizados en jerarquías para proporcionar un nivel alto de partición.

PRODUCTOS A OBTENER:

1. Diagramas de estructura.

Actividades específicas

1. Organizar el paquete de subsistemas

i. En el explorador de Rhapsody, con el botón derecho del mouse presionar sobre el paquete de subsistemas y seleccionar *AddNew* > *Package*.

ii. Renombrar el paquete de acuerdo al diseño de los subsistemas que componen al sistema.

> Al nombrar los subsistemas se debe respetar la notación *Nombre_Subsystem*, donde *Nombre* regularmente es una abreviación de un nombre más largo (por ejemplo, para el subsistema (Gestión de Conexión), el subsistema tendría el nombre GC_Subsystem.

iii. Es necesario presionar *OK* para terminar la organización del paquete.

iv. Será necesario repetir estos pasos de acuerdo al número de subsistemas en los que se vaya a dividir la funcionalidad completa del sistema.

2. Organizar los elementos entre paquetes

i. En el explorador de Rhapsody, expandir el paquete de Arquitectura y la categoría de objetos.

ii. Seleccionar los diagramas con estereotipo <<*Subsystem*>> y arrastrarlos hacia la categoría objetos de los paquetes (subsistemas) que fueron creados en la actividad específica 1.

> Los objetos son eliminados del paquete de Arquitectura y agregados a los paquetes de Subsistemas.

5.3.2.CREAR LA ARQUITECTURA DE LOS SUBSISTEMAS

Diseñar el modelo de objetos de la arquitectura para mostrar las relaciones entre los elementos a nivel de subsistema

El diagrama del modelo de objetos para la arquitectura de los subsistemas identifica como se encuentran interconectados los componentes del sistema a este nivel de descomposición. Este diagrama mostrará la realización de los flujos entre los objetos mediante los vínculos y los puertos. Los flujos son utilizados para un análisis de alto nivel, y los vínculos son utilizados para la ejecución o realización.

5.3. 2.1. Diseñar el diagrama del modelo de objetos

Especificar la estructura de las clases, objetos, e interfaces en el sistema y las relaciones estáticas que existen entre estos

Los diagramas del modelo de objetos proporcionan una representación gráfica de la estructura del sistema.

Estos diagramas muestran los tipos de objetos en el sistema, los atributos y las operaciones que pertenecen a esos objetos, y las relaciones estáticas que pueden existir entre las clases (tipos).

PRODUCTOS A OBTENER:

1. Diagrama del modelo de objetos (DOBJ).

Actividades específicas

1. Establecer la configuración del diagrama del modelo de objetos

i. En el explorador de Rhapsody, expandir la categoría de Paquetes y seleccionar el paquete de Subsistemas.

ii. Seleccionar *Add New* > *Object Model Diagram* del menú contextual del paquete.

iii. Nombrar el diagrama como "Arquitectura de Subsistemas".

2. Establecer la configuración del diagrama del modelo de objetos

i. En el explorador de Rhapsody, expandir la categoría de Paquetes y, de la categoría *Objects*, seleccionar uno a uno los paquetes de Subsistemas creados en el Objetivo 1.

ii. Arrastrar los objetos con estereotipo <<*Subsystem*>> al área de dibujo para el diagrama del modelo de objetos.

> Todos los objetos con estereotipo <<*Subsystem*>> deben ser arrastrados al modelo de objetos.

iii. Para modelar con mayor facilidad, ajustar las siguientes características para todos los objetos arrastrados al diagrama:

- Con el botón derecho del mouse, presionar sobre el objeto arrastrado y seleccionar *Display Options*.
- En la pestaña General, en el grupo *Display Name*, seleccionar la opción *Name Only*.
- Es necesario presionar *OK* para guardar la configuración del diagrama.

3. Dibujar el resto de los objetos

Es posible que el modelo de objetos contenga a los actores definidos en los diagramas de estructura (al menos aquellos que interactúan con el sistema a este nivel de diseño). De ser así, estos actores deberán agregarse al modelo en esta fase del diseño.

4. Dibujar los vínculos entre los objetos

Los vínculos mostrarán el flujo de información en la arquitectura de subsistemas. Un vínculo en una instancia de una asociación.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Enlace* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el puerto origen y presionar de nuevo el botón sobre el puerto destino.

iii. Es necesario repetir estos pasos para todos los vínculos que se desee introducir.

5.3.2.2. GENERAR EL CÓDIGO Y COMPILAR EL MODELO

Probar el modelo de forma incremental mediante su ejecución.

Las pruebas incrementales permiten la animación de piezas del modelo de acuerdo a como se va desarrollando. Esto da la oportunidad de determinar si el modelo satisface los requisitos o no, y permite la detección temprana de errores. De esta forma, el modelo puede probarse completamente.

En cada iteración que se avance sobre el desarrollo del sistema puede hacerse una validación, así al final se habrá realizado una validación completa del mismo.

PRODUCTOS A OBTENER:

1. Diagrama del modelo de objetos (DOBJ).

Actividades específicas

Un componente de validación es un subsistema físico en forma de librería o código ejecutable que juega un papel importante en la modelación de sistemas grandes que contienen muchas librerías y programas ejecutables. Cada componente contiene categorías de configuración y de especificación de archivos, que son utilizadas para generar, construir y correr un modelo ejecutable. Cada proyecto contiene un componente por default, llamado *DefaultComponent*. Es posible utilizar el componente por default o crear un nuevo componente.

i. En el explorador de Rhapsody, expandir la categoría *Components*.

ii. Presionar dos veces con el botón izquierdo del mouse sobre el *Componente* por *Default* para editar sus características.

iii. Renombrar el componente como "*Simulate*".

iv. Es necesario presionar *Apply* para guardar la configuración del componente. Intente no cerrar la ventana *Features*.

2. Establecer las características del componente

i. Si la ventana *Features* fue cerrada será necesario abrirla de nuevo.

ii. En la pestaña de General, en el grupo *Type*, seleccionar la opción *Executable* si no estuviera ya seleccionada.

iii. En la pestaña *Scope*:

a. Seleccionar la opción *Selected Elements*.
b. Seleccionar los paquetes para los cuales se generará el código (*AnalysisPkg*, *ArchitecturePkg*, *Subsystems Pkg*).

No se selecciona el paquete *RequirementsPkg* puesto que por su naturaleza no existe código ejecutable que generar.

iv. Es necesario presionar OK para terminar la edición de las características del componente.

3. Crear una configuración para el componente de validación

Un componente de validación puede contener muchas configuraciones. Una configuración especifica cómo será producido el componente.

Cada componente contiene una configuración por default, llamada *DefaultConfig*.

i. En el explorador de Rhapsody, expandir el componente *Simulate* y la categoría *Configurations*.

ii. Presionar dos veces con el botón izquierdo del mouse sobre la Configuración por Default para editar sus características.

iii. Renombrar la configuración como "*Debug*".

iv. Es necesario presionar *OK* para guardar la configuración de la configuración.

4. Generar el código fuente

Antes de la generación de código es importante verificar que se ha escogido la configuración activa.

La configuración activa es la configuración para la cual se genera el código fuente. La configuración activa aparece en el menú desplegable de la opción *Code* en la barra de herramientas del proyecto.

i. En el explorador de Rhapsody, con el botón derecho del mouse presionar sobre la configuración *Debug* y seleccionar *Set as Active*.

ii. Seleccionar *Code>Generate>Debug*.

Rhapsody puede desplegar un mensaje para indicar que el directorio *Debug* no existe todavía y solicita permiso para crearlo, se deberá presionar *Yes* para autorizar la acción.

iii. Identificar el tipo de mensaje creado durante la generación de código.

La generación de código puede crear mensajes de dos tipos en la ventana de salida, por ejemplo:

- Si se reciben mensajes de error, se debe presionar dos veces con el botón izquierdo del mouse sobre el error para identificar su origen.
- Si se reciben mensajes de advertencia, se debe presionar dos veces con el botón izquierdo del mouse sobre la advertencia para identificar su origen. Es posible que algunas advertencias surjan porque el modelo todavía no está terminado, en este caso este tipo de advertencias pueden ser ignoradas.

Figura 5.5. Componentes utilizados para generar el componente y simularlo

5. Construir el modelo

Seleccionar *Code>Build Simulate.exe* del menú de tareas o utilizar el icono *Make* de la barra de herramientas.

Para una construcción exitosa del modelo, la ventana de salida muestra el mensaje *Build Done*. Es posible que se creen mensajes de error durante la creación del modelo, se deberá seguir la práctica iii de la actividad específica 4 para removerlos.

5.3. CREAR LOS DIAGRAMAS DE SECUENCIA

Diseñar los diagramas que definen la forma en que se comunican los elementos estructurales de la arquitectura en el tiempo.

Los diagramas de secuencia pueden utilizarse en diferentes niveles de abstracción. A mayores niveles de abstracción, los diagramas muestran las interacciones entre los actores, casos de uso, y objetos. A niveles inferiores de abstracción y para la implementación, los diagramas muestran la comunicación entre las clases y los objetos.

5.3.1. DISEÑAR LOS DIAGRAMAS DE SECUENCIA

Diseñar los diagramas de secuencia que permitan identificar flujos de comunicación entre los componentes de la arquitectura propuesta

Un diagrama de secuencia regularmente muestra la forma en que el sistema interactúa con un escenario a través del envío de mensajes entre las entidades que lo forman.

PRODUCTOS A OBTENER:

1. Diagramas de secuencia (Artefacto DSEC).

Actividades específicas

1. Configurar un nuevo diagrama de secuencia

i. En el explorador de Rhapsody, con el botón derecho del mouse presionar sobre el paquete de Subsistemas (SubsystemPkg) y seleccionar *Add New>Sequence Diagram*.

ii. Renombrar el diagrama de secuencia de acuerdo a la funcionalidad que será probada.

iii. Seleccionar *Design* del grupo de opciones *Operation Mode*.

Los diagramas de secuencia pueden crearse en dos modos:
- En el modo de análisis, es posible dibujar diagramas de secuencia sin agregar elementos al modelo. Es decir se puede hacer una lluvia de ideas sobre el análisis y el diseño sin afectar el código fuente generado.
- En el modo de diseño, cada línea y mensaje creado o renombrado puede realizarse como un elemento (clase, objeto, operación, o evento) será parte del modelo y del código generado.

iv. Es necesario presionar OK para guardar la configuración del diagrama.

Rhapsody crea automáticamente la categoría de Diagramas de Secuencia en el paquete de Subsistemas y agrega el nombre al diagrama creado. Además, Rhapsody habilita la nueva área de dibujo para el diagrama de secuencia.

2. Dibujar las líneas de los actores

Las líneas de los actores muestran como participan tales actores en el escenario. Los actores son representados como líneas de instancia con una columna de líneas diagonales.

> En los diagramas de casos de uso y diagramas de secuencia, los actores describen los elementos externos con los cuales interactúa el contexto del sistema.

i. En el explorador de Rhapsody, expandir el paquete *ArchitecturePkg* y la categoría *Object*.

ii. Seleccionar y arrastrar los actores al diagrama de secuencia.

Consultar el área de Especificación de Requisitos para mayor información sobre la creación y adición de actores al modelo.

> También es posible que un diagrama de secuencia inicie con una frontera del sistema. La frontera del sistema representa el entorno y es mostrada como una columna de pequeñas líneas diagonales. Aquellos eventos u operaciones que no provienen de las líneas de instancia pueden provenir de la frontera del sistema. La frontera del sistema es agregada al diagrama de la misma forma que un actor, a través del icono *Frontera del Sistema*.

3. Dibujar los roles clasificadores

Los roles clasificadores o líneas de instancia son líneas de tiempo etiquetadas con el nombre de una instancia, la cual indica el ciclo de vida de los clasificadores u objetos que participan en el escenario. Estos roles representan una instancia típica en el escenario que está siendo descrito.

> Los roles clasificadores pueden recibir o enviar mensajes a otras líneas de instancia. Cabe mencionar que, en este tipo de diagramas, el tiempo avanza hacia abajo en la dirección del eje vertical.

i. En el explorador de Rhapsody, expandir el paquete *SubsystemsPkg* y la categoría Packages. Identificar el subsistema(s) que formará(n) parte del diagrama y expandir la categoría *Objects*.

> Es importante mencionar que es posible que no todos los subsistemas formen parte de un diagrama de secuencia. De acuerdo a la lógica de cualquier sistema, una visión general de funcionamiento deberá incluir a todos los subsistemas definidos en el diseño inicial. Sin embargo, y de acuerdo al aumento de granuralidad en el diseño, es posible establecer un escenario donde interactúen únicamente un par de subsistemas, por ejemplo.

ii. Presionar sobre el subsistema (en la categoría *Objects*) y arrastrarlo hacia el diagrama.

Rhapsody crea el rol clasificador con el nombre del rol en el panel de Nombres.

iii. Será necesario repetir estos pasos hasta que todos los subsistemas hayan sido integrados al escenario.

4. Dibujar los mensajes

Un mensaje representa una interacción entre los objetos, o entre un objeto y el entorno. Un mensaje puede ser un evento, una operación disparada, o una operación primitiva.

Dependiendo la forma de línea, los mensajes pueden ser interpretados de la siguiente forma:

- Si el mensaje es una línea horizontal, el mensaje es interpretado como una operación disparada si el objetivo es una clase reactiva, o una operación primitiva si el objetivo es una clase no reactiva. Un mensaje que es una línea horizontal indica que la operación es síncrona.
- Si el mensaje es una línea inclinada, el mensaje es interpretado como un evento si el objetivo es una clase reactiva, o como una operación primitiva si el objetivo es una clase no reactiva. Un mensaje que es una línea inclinada enfatiza que el tiempo transcurre entre el envío y la recepción de los mensajes. Los mensajes que son líneas inclinadas pueden atravesar a otras.
- Si el mensaje o la línea del mensaje regresa así misma, el mensaje es interpretado como una operación primitiva si la flecha regresa a una clase no reactiva o si la flecha regresa inmediatamente; o es interpretada como un evento si la flecha regresa algún tiempo después. La flecha puede dirigirse hacia cualquier lado de la línea de instancia.

Las clases reactivas pueden recibir eventos, operaciones disparadas, y operaciones primitivas. Las clases no reactivas pueden recibir solamente mensajes que son llamadas a las operaciones primitivas.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Mensaje* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el actor del cual saldrá el mensaje.

iii. Con el botón izquierdo del mouse, presionar sobre el rol clasificador o actor que recibirá el mensaje.

Rhapsody crea un mensaje con el nombre por default de *event_n*( ), donde n es un entero incremental que inicia en 0.

iv. Renombrar el mensaje de acuerdo al escenario que se está describiendo.

Dado que se está creando un diagrama de secuencia en el modo de diseño, cada que sea agregado un nuevo mensaje Rhapsody preguntará si se desea realizar dicho mensaje. Será necesario seleccionar OK para realizar cada mensaje agregado al diagrama. Rhapsody agregará también los mensajes a la categoría *Events* de cada subsistema relacionado en el diagrama.

5. Dibujar una ocurrencia de interacción

Una ocurrencia de interacción (o diagrama de secuencia de referencia) permite hacer referencia a otro diagrama de secuencia desde un diagrama inicial. Esto permite la descomposición de escenarios complejos en escenarios más pequeños que pueden ser reutilizados. No es una tarea obligada, pero simplifica el diseño.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Ocurrencia* que está en la barra de dibujo.

ii. Con el botón izquierdo del mouse, trazar la ocurrencia entre las líneas de instancias y/o mensajes que requieren de dicha referencia.

La ocurrencia de interacción aparece como una caja con la etiqueta Ref en la esquina superior izquierda.

iii. Renombrar la ocurrencia de acuerdo a su objetivo en el diagrama.

El nuevo diagrama deberá ser creado siguiendo los pasos anteriores a partir de la ocurrencia de interacción (Con el botón derecho del mouse, presionar sobre *Create Reference Sequence Diagram*).

6. Dibujar los intervalos de tiempo

Los diagramas de secuencia pueden especificar la cantidad máxima de tiempo que puede transcurrir entre dos puntos. Un intervalo de tiempo es una anotación vertical que muestra cuanto tiempo (real) debe transcurrir entre dos puntos en el escenario.

El nombre del intervalo de tiempo no está restringido a ser únicamente un número o unidad, se puede introducir libremente cualquier texto.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Intervalo de Tiempo* que está en la barra de dibujo.

ii.Con el botón izquierdo del mouse, presionar sobre la línea de instancia en donde comenzará el intervalo y arrastrar hacia donde terminará. Al presionar el botón izquierdo de nuevo, el intervalo quedará dibujado.

Rhapsody dibuja dos líneas horizontales (en los puntos de inicio y fin) y dos puntas de flecha en el centro que indican el tiempo transcurrido entre los dos puntos.

iii. Renombrar la etiqueta del intervalo de tiempo.

7. Establecer las características de tipo y retorno de los mensajes

Es posible definir características especiales para cada mensaje que interactúa en el escenario.

i. En el explorador de Rhapsody, presionar dos veces con el botón izquierdo del mouse sobre el mensaje que se quiera modificar (expandir el paquete del Subsistema relacionado con el mensaje, expandir la categoría *Object*, y expandir la parte (rol clasificador) asociado con el mensaje).

ii. En la pestaña General, en el grupo *Returns*, desactivar la casilla de verificación '*Use existing type*' y en la caja de 'C++ *declaration*' introducir el tipo de mensaje utilizado (p.e. *bool*, *integer*, *double*, etc.).

iii. En la pestaña Implementation, introducir el tipo de retorno (p.e. *void*, *integer*, *TRUE*, *FALSE*, etc.).

iv. Es necesario presionar *OK* para guardar las características del mensaje.

5.3.2. VALIDAR LOS DIAGRAMAS DE SECUENCIA

Probar los diagramas de secuencia en entornos de simulación controlada

De acuerdo a como el modelo se vuelve cada vez más complicado, se recomienda detenerse y validarlo periódicamente y proporcionar una depuración a nivel de diseño. Uno de los métodos primarios más usados consiste en simular el modelo, una animación.

PRODUCTOS A OBTENER:

1. Validaciones del modelo.

Actividades específicas

1. Configurar las propiedades para la animación de los diagramas de secuencia

De acuerdo a como el modelo se vuelva cada vez más complicado, se recomienda detenerse y validarlo periódicamente y proporcionar una depuración a nivel de diseño. Uno de los métodos primarios más usados consiste en simular el modelo, una animación. Una animación es la ejecución observable del comportamiento y definiciones asociadas al modelo que se realiza mediante la ejecución del código generado, con la instrumentación para las clases, operaciones y asociaciones.

i. En el explorador de Rhapsody, con el botín derecho del mouse presionar sobre la configuración definida (Debug por default) para abrir la ventana de propiedades.

ii. En la pestaña de *Initialization*, asegurarse de establecer los siguientes valores:
- En el grupo de *Initial Instances*, seleccionar la opción '*Explicit*'.
- Seleccionar la casilla de comprobación de '*Generate Code for Actors*'.

iii. Definir el entorno para la creación apropiada del archivo ejecutable. En la pestaña *Settings*, establecer los siguientes valores:
- En el grupo de *Instrumentation*, en la lista desplegable para *Instrumentation Mode*, seleccionar '*Animation*'. Esto agregará código de instrumentación que hace posible la animación del modelo.
- En el grupo de *Time Model*, seleccionar la opción '*Real*' (para tiempo real).
- En el grupo de *Statechart Implementation*, seleccionar la opción '*Flat*'. Rhapsody implementará los estados como simples variables de tipo enumerado.

2. Regenerar el código y compilar el modelo

i. Confirmar que *Debug* es la configuración actual (si es que no se cambió el nombre de la configuración anteriormente). Si es necesario, en el explorador de Rhapsody con el botón derecho del mouse presionar sobre la configuración y seleccionar '*Set as Active Configuration*'.

ii. Limpiar la ventana de salida de la compilación mediante la opción '*Clear*' que aparece al presionar el botón derecho del mouse sobre la misma.

iii. *Seleccionar Code>Regenerate>Debug*. Si es necesario, corregir los errores que surjan del proceso.

iv. *Seleccionar Code>Rebuild Simulate.exe*. Si es necesario, corregir los errores que surjan en el proceso.

Figura 5.6. Componentes utilizados para iniciar simulación y para dibujar las swimlanes

3. Iniciar la animación

i. Seleccionar *Code>Run* de la barra de tareas o presionar con el botón izquierdo del mouse el icono *Run*.

4. Animar el diagrama de secuencia

Los diagramas de secuencia animados muestran la forma en que los objetos transmiten los mensajes mientras el modelo es ejecutado.

Es decir, es posible observar la comunicación en la ejecución del sistema para después comparar la secuencia de mensajes con diagramas de secuencia no animados y observar si el modelo se está comportando correctamente.

i. Seleccionar *Tools>Animated Sequence Diagram* de la barra de herramientas.

ii. En la caja de dialogo *Open Sequence Diagram*, expandir el paquete *SubsystemsPkg* y seleccionar el diagrama que será animado.

iii. Es necesario presionar *OK* para crear la versión animada del diagrama.

iv. Con el botón izquierdo del mouse, presionar sobre el icono *Animation* que está en la barra de tareas.

5. Salir de la animación

i. Con el botón derecho del mouse, presionar sobre el icono *Animación*. En la barra de tareas de *Animation* presionar el botón *Animación* .

ii. Es necesario seleccionar *Yes* para confirmar la terminación de la animación.

5.3.4. CREAR LOS DIAGRAMAS DE ACTIVIDADES

Diseñar los diagramas que reflejen el aspecto dinámico del sistema y el flujo de control entre las actividades

Los diagramas de actividades describen las interacciones esenciales entre el sistema y su entorno, y las interconexiones de los comportamientos que son responsabilidad de los subsistemas o componentes. Estos diagramas también pueden usarse para modelar una operación o los detalles de un cómputo.

5.3.4.1. DISEÑAR LOS DIAGRAMAS DE ACTIVIDADES

Diseñar los diagramas de actividades que muestren el flujo funcional que soporta la estructura de subsistemas o componentes

Un diagrama de actividades regularmente muestra la forma en que el sistema interactúa con un escenario y con los componentes que lo forman a través de flujos funcionales.

PRODUCTOS A OBTENER:

1. Diagramas de actividades (Artefacto DACT).

Actividades específicas

1. Configurar un nuevo diagrama de actividades

i. En el explorador de Rhapsody, expandir el paquete *SubsystemPkg*, expandir el subsistema para el cual se va a realizar el diagrama de actividades, expandir el objeto sobre el cual se diseñar el diagrama de actividades, y por último la categoría *Parts*. Con el botón izquierdo del mouse, presionar sobre la parte que se trabajará y seleccionar *Add New>Activity Diagram*. Es importante mencionar que esta actividad deberá realizarse para todos los subsistemas en que se dividió el sistema.

Rhapsody agrega automáticamente la categoría de *Activity Diagram* y el nuevo diagrama de actividades para la parte escogida en el explorador, y abre una nueva área de dibujo.

2. Dibujar las swimlanes

Las *swimlanes* organizan a los diagramas de actividades en secciones de responsabilidad para las acciones y subacciones. Las líneas verticales dividen cada *swimlane* de otros adyacentes.

Es posible que sea conveniente expandir el área de dibujo mediante el cierre del explorador de Rhapsody. Esta acción proporciona más espacio para la creación de los diagramas.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Swimlanes* que se encuentra en la barra de dibujo, y colocarlo en la zona de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el icono *Divisor de Swimlanes* que se encuentra en la barra de dibujo. Para cada *swimlane*:

- Con el botón izquierdo del mouse, presionar en el área del *Swimlanes* en donde se desea hacer la división.

iii. Con el botón derecho del mouse, presionar sobre el *swimlane* para abrir la ventana de propiedades y cambiar de nombre de acuerdo a la funcionalidad que se representará.

3. Dibujar los elementos de acción

Un elemento de acción representa invocaciones de una función con una sola transición de salida cuando la función termina.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Acción* que se encuentra en la barra de dibujo, presionar de nuevo el botón izquierdo del mouse para colocarlo en la zona de dibujo y presionar Ctrl+Enter.

Si se presiona únicamente Enter, se estará moviendo el curso a una nueva línea para la captura de texto dentro de la acción. Con Ctrl+Enter se indica que el proceso ha sido terminado.

ii. Con el botón derecho del mouse, presionar sobre el elemento de acción y seleccionar *Features*.

iii. En la pestaña de General, en el campo *Name*, se deberá introducir el nombre deseado para la acción.

iv. En la pestaña de *Description*, se deberá introducir una descripción exacta de la acción.

La descripción de los elementos de acción es muy importante para la documentación del sistema.

v. Es necesario presionar *OK* para guardar los cambios y salir de la edición de las propiedades de la acción.

vi. Con el botón derecho del mouse, presionar sobre el elemento de acción y seleccionar *Display Options*:

- En el grupo *Show Name*, seleccionar la opción *Name*.
- En el grupo *Show Action*, *Description* or *Label*, seleccionar la opción *Description*.
- Es necesario presionar *OK* para guardar los cambios y salir de la edición de las propiedades de visualización.

vii. Repetir todos los pasos anteriores para agregar al modelo tantos elementos de acción como sea necesario.

4. Dibujar el flujo de inicio

El elemento de acción con el flujo de inicio es el elemento de acción por default. Es decir, es el elemento de acción inicial del diagrama que espera peticiones para ejecutarse. Una vez que la acción por default es activada, otras acciones en el diagrama pueden realizarse.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Flujo por Default* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el borde derecho del elemento de acción para fijar el flujo de inicio.

5. Dibujar las subactividades

Una subactividad representa la ejecución de una secuencia de pasos anidada dentro de otra actividad.

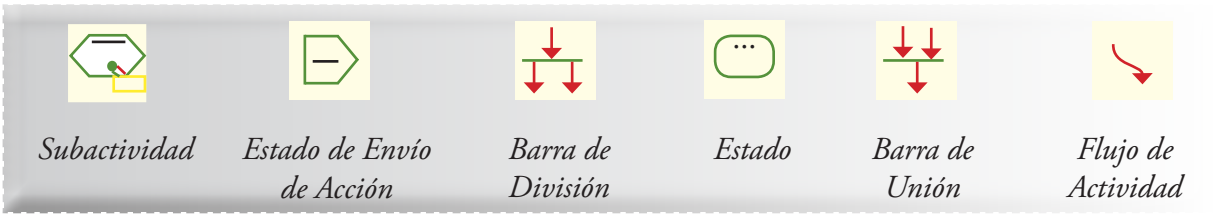


Figura 5.7. Componentes utilizados para dibujar los diagramas de actividades y de estados

i. Con el botón izquierdo del mouse, presionar sobre el icono *Subactividad* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar dentro del *swimlane* en donde será agregada la subactividad.

iii. Con el botón derecho del mouse, presionar sobre la subactividad agregada y seleccionar *Features*.

iv. En la pestaña de General, en el campo *Name*, se deberá introducir el nombre deseado para la subactividad.

v. Es necesario presionar *OK* para guardar los cambios y salir de la edición de las propiedades de la subactividad.

6. Dibujar los estados de envío

Los estados de envío pueden utilizarse para representar el envío de eventos a entidades externas. Estos estados permiten la especificación del evento a enviar, el evento objetivo, y los valores para los argumentos del evento.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Estado de Envío de Acción* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar dentro del *swimlane* en donde será agregado el estado de envío.

iii. Con el botón izquierdo del mouse, presionar dos veces sobre el estado de envío agregado para abrir la caja de *Features*.

iv. En la pestaña de General, en la lista desplegable para *Target*, seleccionar el elemento a donde será enviado el evento.

v. En la lista desplegable para *Event*, seleccionar el evento que será enviado. En caso de que se tratase de un evento nuevo, seleccionar <<*New*>> y dar un nombre al evento creado.

vi. Es necesario presionar *OK* para guardar los cambios y salir de la edición de propiedades del estado de envío.

vii. Será necesario repetir esta actividad de acuerdo al número de estados de envío que se desee introducir.

Figura 5.8. Componentes utilizados para dibujar las transiciones

7. Dibujar las transiciones

Una transición representa una relación entre dos estados que indica que un objeto en el primer estado realizará ciertas acciones y entrará al segundo estado cuando un evento específico ocurra y ciertas condiciones específicas sean cubiertas.

Las transiciones pueden ser de los siguientes tipos:

- Transiciones simples entre las acciones, las cuales indican un único flujo de comunicación entre acciones.
- Transiciones *Fork* y *Join*, una transición *Fork* representa la división de un solo flujo en dos o más flujos salientes; mientras que una transición *Join* representa la mezcla de dos o más flujos entrantes en un solo flujo de salida.
- Transiciones de descanso, las cuales representan una interrupción de tiempo en el envío de un evento.

i. Para dibujar las transiciones entre las acciones:

- Con el botón izquierdo del mouse, presionar sobre el icono *Flujo de Actividad* que se encuentra en la barra de dibujo.
- Con el botón izquierdo del mouse, presionar sobre la subactividad origen y presionar de nuevo al llegar a la subactividad destino.
- Introducir el nombre deseado y presionar Ctrl+Enter.

Es posible cambiar el tipo de línea de una transición si se presiona el botón derecho del mouse sobre la transición y se selecciona de *Line Shape* los tipos *Straight*, *Spline*, *Rectilinear*, o *Re-Route*.

- Para etiquetar las transiciones es necesario presionar el botón derecho del mouse sobre la transición y seleccionar *Features*. Con el botón izquierdo del mouse, presionar sobre el botón **L** que está al lado de *Name* e introducir la etiqueta deseada.
- Es necesario presionar *OK* para guardar los cambios y salir de la edición de propiedades de la transición.
- Será necesario repetir esta actividad de acuerdo al número de transiciones que se desee introducir.

ii. Para dibujar las transiciones Fork:
- Con el botón izquierdo del mouse, presionar sobre el icono *Barra de División* que se encuentra en la barra de dibujo.
- Con el botón izquierdo del mouse, presionar sobre el área de dibujo para colocar la barra *Fork* entre las actividades donde se desee usar.
- Dibujar las transiciones utilizando la actividad 7. Cabe recordar que la transición *Fork* solo podrá tener una transición de entrada.
- Introducir el nombre deseado para cada transición y presionar Ctrl+Enter.

iii. Para dibujar las transiciones Join:
- Con el botón izquierdo del mouse, presionar sobre el icono *Barra de Unión* que se encuentra en la barra de dibujo.

- Con el botón izquierdo del mouse, presionar sobre el área de dibujo para colocar la barra *Join* entre las actividades donde se desee usar.
- Dibujar las transiciones utilizando la actividad 7. Cabe recordar que la transición *Join* solo podrá tener una transición de salida.
- Introducir el nombre deseado para cada transición y presionar Ctrl+Enter.

iv. Para dibujar las transiciones de descanso:
- Dibujar una transición entre actividades, tal y como se indico en la actividad 7 pero considerando que el origen y el destino sean la misma actividad.
- Introducir el tiempo del retardo en el formato tm(tiempo) y presionar Ctrl+Enter.

8. Especificar acciones sobre una transición

i. Con el botón derecho del mouse, presionar sobre la transición para abrir la ventana de propiedades.

ii. En la pestaña de General, en la ventana de *Action*, introducir el código que se pretende asociar a la transición (p.e., locationUpdate( );)

iii. Es necesario presionar OK para aplicar los cambios y salir de la edición de propiedades. El nombre del comando será agregado al nombre de la transición.

9. Crear los diagramas de subactividades

Después de haber realizado la actividad 5, es necesario crear el diagrama dentro de la subactividad siguiendo todos los pasos anteriores.

i. Con el botón derecho del mouse, presionar sobre el subdiagrama dibujado anteriormente y seleccionar *Open Sub Activity Diagram*.

ii. Dibujar los elementos de acción, flujo de inicio, estados de envío y transiciones necesarias para el nuevo subdiagrama.

iii. Es necesario presionar OK para aplicar los cambios y salir de la edición de propiedades. El nombre del comando será agregado al nombre de la transición.

9. Crear los diagramas de subactividades

Después de haber realizado la actividad 5, es necesario crear el diagrama dentro de la subactividad siguiendo todos los pasos anteriores.

i. Con el botón derecho del mouse, presionar sobre el subdiagrama dibujado anteriormente y seleccionar *Open Sub Activity Diagram*.

ii. Dibujar los elementos de acción, flujo de inicio, estados de envío y transiciones necesarias para el nuevo subdiagrama.

iii.Será necesario repetir estas actividades de acuerdo al número de subsistemas en que se haya dividido el sistema.

v. Es necesario presionar OK para guardar las características del mensaje.

5.3.4.2. VALIDAR LOS DIAGRAMAS DE ACTIVIDADES

Probar los diagramas de actividades en entornos de simulación controlada

Tal y como se mencionó anteriormente, de acuerdo a como el modelo se vuelve más complicado, se recomienda detenerse y validarlo periódicamente y proporcionar una depuración a nivel de diseño mediante una animación.

PRODUCTOS A OBTENER:

1. Validaciones del modelo

Actividades específicas

1. Regenerar el código y compilar el modelo

i. Confirmar que *Debug* es la configuración actual (si es que no se cambió el nombre de la configuración anteriormente). Si es necesario, en el explorador de Rhapsody con el botón derecho del mouse presionar sobre la configuración y seleccionar '*Set as Active Configuration*'.

ii. Limpiar la ventana de salida de la compilación mediante la opción '*Clear*' que aparece al presionar el botón derecho del mouse sobre la misma.

iii. Seleccionar *Code>Regenerate>Debug*. Si es necesario, corregir los errores que surjan del proceso.

iv. Seleccionar *Code>Rebuild Simulate.exe*. Si es necesario, corregir los errores que surjan en el proceso.

2. Iniciar la animación

i. Seleccionar *Code>Run* de la barra de tareas o presionar con el botón izquierdo del mouse el icono *Run*.

3. Animar el diagrama de actividades

Los diagramas de actividades animados muestran la transición de un estado a otro mientras el modelo está en ejecución.

i. Seleccionar *Tools>Animated Activity Diagram* de la barra de herramientas.

ii. En la caja de dialogo *Open Animated Activity Diagram*, seleccionar la instancia para la cual se requiere la animación.

iii. Es necesario presionar *OK* para crear la versión animada del diagrama.

iv. Con el botón izquierdo del mouse, presionar sobre el icono *Animación* que está en la barra de tareas.

4. Salir de la animación

i. Con el botón derecho del mouse, presionar sobre el icono ( ). En la barra de tareas de *Animation* presionar el botón *Salir*.

ii. Es necesario seleccionar *Yes* para confirmar la terminación de la animación.

5.3.5. CREAR LOS DIAGRAMAS DE MÁQUINA DE ESTADO

Diseñar los diagramas que definan el comportamiento de todas las partes de un sistema, incluidos los subsistemas

Los diagramas de máquina de estados definen el comportamiento de los clasificadores (actores, casos de uso, o clases), objetos, incluyendo los estados a los cuales estos pueden entrar durante su ciclo de vida y los mensajes, eventos, u operaciones que causan la transición de estado a estado. Las máquinas de estado son una herramienta clave de animación usada para verificar el flujo funcional.

5.3.5.1. DISEÑAR LOS DIAGRAMAS DE MÁQUINA DE ESTADOS

Diseñar los diagramas de máquina de estados que definan el flujo funcional del comportamiento basado en estados

Un diagrama de máquina de estados es un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores.

PRODUCTOS A OBTENER:

1. Diagramas de máquina de estados (Artefacto DMAQ)

Actividades específicas

1. Configurar un nuevo diagrama de máquina de estados

i. En el explorador de Rhapsody, expandir el paquete *SubsystemPkg*, expandir el subsistema para el cual se va a realizar el diagrama de máquina de estados, expandir el objeto sobre el cual se diseñar el diagrama de máquina de estados, y por último la categoría *Parts*. Con el botón izquierdo del mouse, presionar sobre la parte que se trabajará y seleccionar *Add New>Statechart*. Es importante mencionar que esta actividad deberá realizarse para todos los subsistemas en que se dividió el sistema.

> Rhapsody agrega automáticamente la categoría de Statechart y el nuevo diagrama de actividades para la parte escogida en el explorador, y abre una nueva área de dibujo.

2. Dibujar los estados

Un estado es una representación gráfica de la condición de un objeto. Regularmente refleja un cierto conjunto de datos internos (atributos) y relaciones.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Estado* que se encuentra en la barra de dibujo, y colocarlo en la zona de dibujo. Se creará un estado con un nombre por default, state_n, donde n es un número igual o mayor que 0.

> Es importante considerar que, al igual que los diagramas de actividades, una máquina de estados puede contener otro diagrama de estados más específico. Por lo tanto, el tamaño del estado puede manipularse desde que se está insertando por primera vez.

ii. Introducir el nombre y presionar Enter.

3. Dibujar los estados anidados

i. Con el botón izquierdo del mouse, presionar sobre el icono *Estado* que se encuentra en la barra de dibujo, y colocarlo dentro del estado dibujado anteriormente.

ii. Introducir el nombre y presionar Enter.

iii. Será necesario repetir estas actividades de acuerdo al número de estados que se desee introducir al diagrama anidado.

4. Dibujar el conector de inicio

Uno de los estados del objeto debe ser el estado por default, es decir, el estado en el cual el objeto se encuentra cuando es instanciado por primera vez.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Conector por Default* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el borde derecho del estado para fijar el conector de inicio.

Es importante mencionar que tanto el diagrama de primer nivel así como el resto de diagramas anidados, deberán tener un conector de inicio.

5. Dibujar los estados de envío

Tal y como se mencionó anteriormente para los diagramas de actividades, los estados de envío pueden utilizarse para representar el envío de eventos a entidades externas. Estos estados permiten la especificación del evento a enviar, el evento objetivo, y los valores para los argumentos del evento.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Estado de Envío de Acción* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el lugar en donde será agregado el estado de envío.

iii. Con el botón izquierdo del mouse, presionar dos veces sobre el estado de envío agregado para abrir la caja de *Features*.

iv. En la pestaña de General, en la lista desplegable para *Target*, seleccionar el elemento a donde será enviado el evento.

v. En la lista desplegable para *Event*, seleccionar el evento que será enviado. En caso de que se tratase de un evento nuevo, seleccionar <<*New*>> y dar un nombre al evento creado.

vi. Es necesario presionar *OK* para guardar los cambios y salir de la edición de propiedades del estado de envío.

vii. Será necesario repetir esta actividad de acuerdo al número de estados de envío que se desee introducir.

6. Dibujar las transiciones

Una transición representa la respuesta a un mensaje en un estado dado.

i. Con el botón izquierdo del mouse, presionar sobre el icono *Transición* que se encuentra en la barra de dibujo.

ii. Con el botón izquierdo del mouse, presionar sobre el estado origen y presionar de nuevo al llegar al estado destino.

iii. Introducir el nombre deseado y presionar Ctrl+Enter.

Es posible cambiar el tipo de línea de una transición si se presiona el botón derecho del mouse sobre la transición y se selecciona de Line Shape los tipos Straight, Spline, Rectilinear, o Re-Route.

Las transiciones para las máquinas de estado también pueden ser del tipo simples, fork, join o descanso, por lo que es posible que sea necesario consultar la actividad 7 del Objetivo 4.1.

5.3.5.2. VALIDAR LOS DIAGRAMAS DE MÁQUINA DE ESTADOS

Probar los diagramas de actividades en entornos de simulación controlada

Tal y como se mencionó anteriormente, de acuerdo a como el modelo se vuelve más complicado, se recomienda detenerse y validarlo periódicamente y proporcionar una depuración a nivel de diseño mediante una animación.

PRODUCTOS A OBTENER:

1. Validaciones del modelo

Actividades específicas

1. Regenerar el código y compilar el modelo

i. Confirmar que *Debug* es la configuración actual (si es que no se cambió el nombre de la configuración anteriormente). Si es necesario, en el explorador de Rhapsody con el botón derecho del mouse presionar sobre la configuración y seleccionar '*Set as Active Configuration*'.

ii. Limpiar la ventana de salida de la compilación mediante la opción '*Clear*' que aparece al presionar el botón derecho del mouse sobre la misma.

iii. Seleccionar *Code>Regenerate>Debug*. Si es necesario, corregir los errores que surjan del proceso.

iv. Seleccionar *Code>Rebuild Simulate.exe*. Si es necesario, corregir los errores que surjan en el proceso.

2. Iniciar la animación

i. Seleccionar *Code>Run* de la barra de tareas o presionar con el botón izquierdo del mouse el icono *Run*.

3. Animar el diagrama de actividades

Los diagramas de actividades animados muestran la transición de un estado a otro mientras el modelo está en ejecución.

i. Seleccionar *Tools>Animated Activity Diagram* de la barra de herramientas.

ii. En la caja de dialogo *Open Animated Activity Diagram*, seleccionar la instancia para la cual se requiere la animación.

iii. Es necesario presionar *OK* para crear la versión animada del diagrama.

iv. Con el botón izquierdo del mouse, presionar sobre el icono *Animación* que está en la barra de tareas.

4. Salir de la animación

i. Con el botón derecho del mouse, presionar sobre el icono *Pausa*. En la barra de tareas de *Animation* presionar el botón *Salir*.

ii. Es necesario seleccionar *Yes* para confirmar la terminación de la animación.

5.4. GESTIÓN DE CONTRATOS

Un área de proceso de la categoría de Soporte de SPIES

PROPÓSITO

El propósito de la Gestión de Contratos es establecer y mantener puertos basados en contratos que permitan la definición de un contrato que especifique las entradas y salidas específicas permisibles de un componente.

NOTAS INTRODUCTORIAS AL ÁREA DE PROCESO

Un puerto basado en contrato puede tener las siguientes interfaces:

- Interfaces proporcionadas que caracterizan las peticiones que el entorno puede realizar. Regularmente las interfaces proporcionadas son denotadas por una notación lollipop.
- Interfaces requeridas que caracterizan las peticiones que se pueden realizar de un puerto a su entorno (objetos externos). Una interfaz requerida se denota por una notación de socket.

Las interfaces proporcionadas y requeridas permiten encapsular elementos del modelo mediante la definición del acceso a través del puerto. La siguiente figura muestra un ejemplo de las interfaces de un puerto.

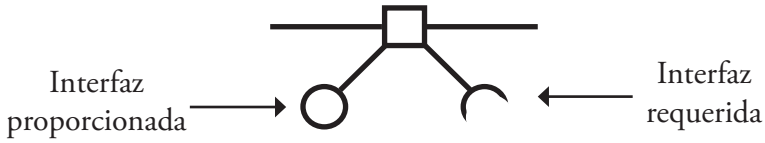


Figura 5.9. Interfaces de un puerto

Los puertos que no están basados en contratos, llamados también puertos rápidos, permiten transmitir mensajes a la parte apropiada de una clase a través de un conector. Inicialmente estos no requieren el establecimiento de un contrato, pero permiten el seguimiento de datos entrantes a través del puerto hacia la parte apropiada.

ÁREAS DE SPIES RELACIONADAS

Consultar el área de Diseño del Producto para mayor información sobre la construcción de modelos estructurados que permitan establecer contratos de comunicación entre los puertos.

Resumen de Actividades

1. Establecer contratos para los puertos
    1.1. Especificar el contrato del puerto
    1.2. Especificar las interfaces del puerto
    1.3. Invertir los contratos

ACTIVIDADES POR OBJETIVO

5.4.1. ESTABLECER CONTRATOS PARA LOS PUERTOS

Se establecen contratos para los puertos y establecer un control sobre la entrada y salida de datos

La definición de contratos para los puertos (o interfaces del sistema) asegura que los datos, tantos requeridos como proporcionados, estén disponibles en tiempo y forma para la ejecución de alguna operación dentro del entorno del sistema.

5.4.1.1. ESPECIFICAR EL CONTRATO DEL PUERTO

Establecer puertos basados en contratos para refinar el diseño del sistema

El desarrollo de las actividades propuestas por SPIES con un ciclo iterativo permite dividir a todo el proyecto en un conjunto interconectado de componentes manejables. Regularmente, cualquier conjunto ordenado de actividades es una estructura orientada al proceso que proporciona un esquema para identificar y organizar las unidades lógicas de trabajo a ser gestionadas. La estructura de trabajo de SPIES proporciona una referencia y un mecanismo organizacional para la asignación del esfuerzo, calendario, y responsabilidad y es usado como el marco subyacente para planificar, organizar y controlar el trabajo realizado en el proyecto.

PRODUCTOS A OBTENER:

1. Puertos basados en contratos

Actividades específicas

1. Con el botón derecho del mouse, presionar sobre el puerto y seleccionar *Features*.
2. En la pestaña Contract seleccionar el tipo de interfaz que le será asociada al puerto: *Provided* o *Required*. Presionar el botón *Add*.
3. Seleccionar <*New*> de la lista desplegable y presionar *OK* para configurar la interfaz.
4. En la pestaña General reemplazar el nombre de la interfaz por uno acorde al diseño.
5. En la pestaña Operations, presionar <*New*> y seleccionar Reception.
6. Introducir un nombre para la operación a realizarse en la interfaz y presionar *OK*.
7. Es necesario repetir los pasos para agregar todos los eventos necesarios para la interfaz creada.

Regularmente se asocia el nombre In para interfaces proporcionadas y *Out* para interfaces requeridas.

Al nombrar las operaciones, Rhapsody emitirá un mensaje que indica que el evento indicado no puede encontrarse. Deberán presionar Yes para crear el nuevo evento. Rhapsody agrega el evento a la pestaña *Operations*.

5.4.1.2. ESPECIFICAR LAS INTERFACES DEL PUERTO

Establecer las interfaces para los puertos que permitan controlar la entrada y salida de datos

El desarrollo de las actividades propuestas por SPIES con un ciclo iterativo permite dividir a todo el proyecto en un conjunto interconectado de componentes manejables.

Regularmente, cualquier conjunto ordenado de actividades es una estructura orientada al proceso que proporciona un esquema para identificar y organizar las unidades lógicas de trabajo a ser gestionadas. La estructura de trabajo de SPIES proporciona una referencia y un mecanismo organizacional para la asignación del esfuerzo, calendario, y responsabilidad y es usado como el marco subyacente para planificar, organizar y controlar el trabajo realizado en el proyecto.

PRODUCTOS A OBTENER:

1. Interfaces de puertos

Actividades específicas

1. **Con el botón derecho del mouse, presionar sobre el puerto y seleccionar Features.**
2. **En la pestaña General, desde la lista desplegable de Contrato, seleccione el contrato tipo In para aquellos puertos que sean del tipo Provided, o Out para aquellos puertos que sean del tipo Required.**

Rhapsody agrega automáticamente las interfaces proporcionadas definidas previamente en In.

3. Es necesario repetir los pasos para establecer, siempre y cuando sea necesario, las interfaces en los puertos.

5.4.1.3. INVERTIR LOS CONTRATOS

Invertir los contratos de los puertos en aquellos elementos que, por el diseño, requieran manejar las entradas y salidas de forma contraria

Es posible que durante el diseño, se determine que sea necesario que las interfaces proporcionadas se conviertan en interfaces requeridas, y viceversa.

PRODUCTOS A OBTENER:

1. Contratos invertidos

Actividades específicas

1. Con el botón derecho del mouse, presionar sobre el puerto al que se desee aplicar el cambio de contrato y seleccionar Features.

2. En la pestaña General, en el grupo Attributes, seleccione la opción Reversed.

3. Es necesario presionar OK para guardar todos los cambios realizados al puerto.

6

RESULTADOS EXPERIMENTALES

Para evaluar la aplicabilidad de SPIES, se han implementando dos prototipos de SE dentro del programa de Maestría en Electrónica y Computación: el sistema Roadruneer (controlador de semáforos), el sistema COYOTE (THE COYOTE UNMANNED AIR VEHICLE SYSTEM) y la tesis denominada "Diseño y desarrollo del sistema de entrenamiento DREAMBLUE utilizando la metodología SPIES". El trabajo que se muestra, es producto del trabajo realizado, en un tiempo no mayor a seis meses, por los alumnos del curso "Metodologías de diseño" de la Maestría en Electrónica y Computación. Por lo que, los equipos desarrolladores recibieron un curso completo sobre el desarrollo de SE e integraron los siguientes equipos:

Roadrunner (Generación 2009-2011):

Ing. José Antonio Juárez Abad.
Ing. Amilcar Díaz Méndez.
Ing. Edgar Manuel Cano Cruz.

COYOTE (Generación 2010-2012):

Ing. Selene Alvarado Legaria.
Ing. Edel Cuevas López.
Ing. Arturo Hernández Méndez.
Ing. Miguel Muñoz Cruz.
Ing. Juan Gabriel Zambrano Nila.
Ing. Héctor Herminio Bayona Acevedo.
Ing. Ayax Aldemar García López.
Ing. Alberto Méndez Jiménez.

TESIS

Ing. Edgar Manuel Cano Cruz.

6.1. SISTEMA ROADRUNNER

El sistema Roadrunner fue el primer sistema diseñado usando SPIES. La metodología SPIES inicia con la *planeación del proyecto* que tiene como propósito establecer una visión común inicial de los objetivos del proyecto, determinar si es viable y reunir la información necesaria para organizar y asignar recursos. Para esto, es necesario que los estudiantes utilicen la plantilla de "Plan de proyecto" del Repositorio de conocimientos (véase Apéndice D). El grupo de trabajo del proyecto Roadrunner no contaba con datos históricos para su planeación. Así, la responsabilidad del estudiante por recoger información y retroalimentarse con la experiencia juega un papel importante en esta fase. Toda la información recogida se actualiza en el repositorio de conocimientos para incrementar el conjunto de prácticas efectivas y aplicarlas en proyectos futuros.

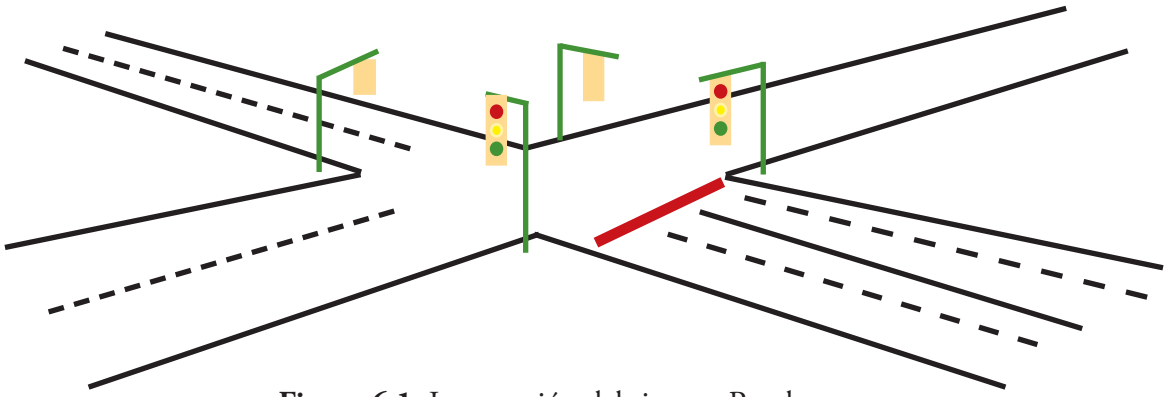


Figura 6.1. Intersección del sistema Roadrunner

6.1.1. ESPECIFICACIÓN DE REQUISITOS DEL SISTEMA ROADRUNNER

La Figura 6.1 muestra una idea general del entorno en el que el sistema Roadrunner debe funcionar. El sistema debe controlar una intersección (caminos 1 y 2) con cuatro semáforos y cruces peatonales. El desarrollo del sistema Roadrunner partió de un conjunto de especificaciones. En el siguiente párrafo se presenta una breve descripción de las especificaciones proporcionadas.

El sistema *Roadrunner Intersection Controller* (RIC) que se debe desarrollar será un control de las intersecciones dentro del campus de la Universidad Tecnológica de la Mixteca y cuyas especificaciones básicas incluyen los siguientes tres componentes: *sensores* (para la detección de vehículos y peatones), *panel frontal con pantalla* (para configurar automáticamente el sistema), y el *componente interno* para gestionar el tráfico. El componente de gestión del tráfico, debe incluir cinco modos: modo de cruce seguro, el modo nocturno de bajo volumen, el modo de ciclo de respuesta, el modo fijo del tiempo de ciclo, y el modo de adaptación. El sensor puede detectar vehículos (vehículos prioritarios y vehículos de emergencia) y peatones.

A partir de las especificaciones del sistema se realizó la fase de *especificación de requisitos* que tiene como propósito definir las propiedades esenciales del sistema que será desarrollado. La especificación de requisitos se compone de dos actividades primarias: captura de requisitos y la definición de casos de uso (que implica usar el Artefacto DREQ para los requisitos y el Artefacto DCUF para los casos de uso).

6.1.2. DIAGRAMA DE CASOS DE USO DEL SISTEMA ROADRUNNER

En base a las especificaciones dadas del Sistema Roadrunner, el equipo de desarrollo identificó los modos operacionales en cruceros, los tipos de detecciones, y las interacciones que el sistema debe soportar. Por lo que se identificaron dos casos de uso primarios: Configuración del Sistema y Administración del Tráfico (véase Figura 6.2). La Configuración del Sistema interactúa ya sea con el operador del panel frontal o el monitor remoto. Independientemente de cómo se realice la configuración, el sistema también realiza la Administración del Tráfico.

Para identificar los casos de uso restantes, el equipo de diseño reviso las especificaciones y definió lo siguiente:

Modos de operación en los cruceros:	Detección de:
• Modo Seguro	• Vehículo
• Modo Nocturno	• Vehículo de emergencia
• Modo Ajustado	• Vehículo con prioridad
• Modo de ciclo de respuesta	• Peatón
• Modo Adaptativo	

La Figura 6.2 muestra el Diagrama de Casos de Uso General del proyecto diseñado con la información identificada por el equipo de desarrollo, de esta manera se describieron los requisitos funcionales y los no funcionales relacionados al Sistema Roadrunner. El artefacto común a esta actividad se encuentra en la base de conocimiento del Apéndice D (Artefacto DCUF). Para construir los DCU el equipo siguió los pasos propuestos por la metodología en la Página 93.

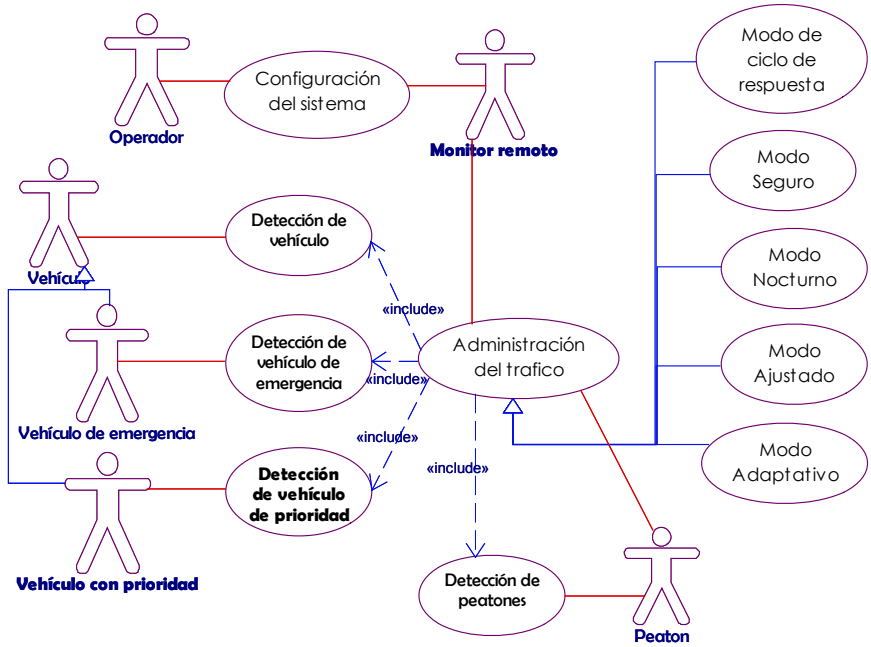


Figura 6.2. Identificación de los Casos de Uso de Roadrunner con SPIES y modelados con Rhapsody

Los Casos de Uso pueden descomponerse en casos de usos más pequeños que especifiquen lo que se tiene que hacer para alcanzar la funcionalidad. Así por ejemplo, para la configuración del sistema se identifico que el sistema debe:

- Controlar los sensores para la detección de vehículos (para ambos caminos).
- Control el sensor para peatones (para ambos caminos).
- Controlar el panel frontal.
- Controlar las luces (para ambos caminos).
- Controlar las luces para peatones (para ambos caminos).

A partir de la información obtenida se genero el DCU de la Figura 6.3., diagrama de uso específico para el CU configuración del sistema (véase Figura 6.2.). Por lo tanto, el equipo separó cada caso de uso y lo desarrolló de forma jerarquizada (véase Figura 6.4). El caso del panel frontal especifica a detalle los requisitos necesarios para cubrir el objetivo del panel.

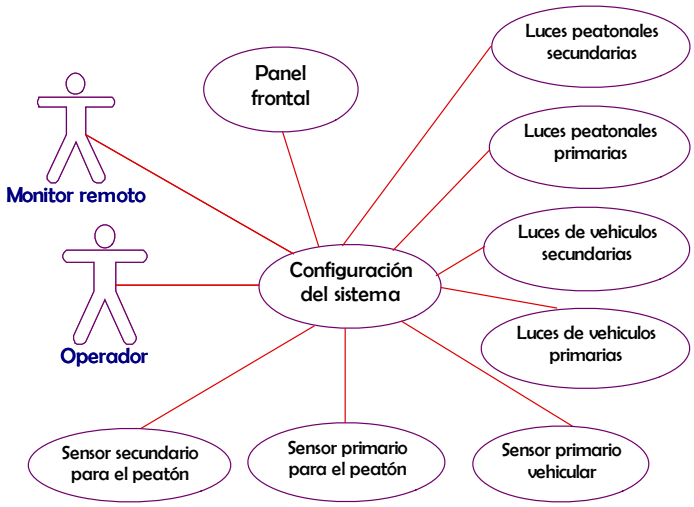


Figura 6.3. Casos de Uso *configuración de sistema* modelados con Rhapsody

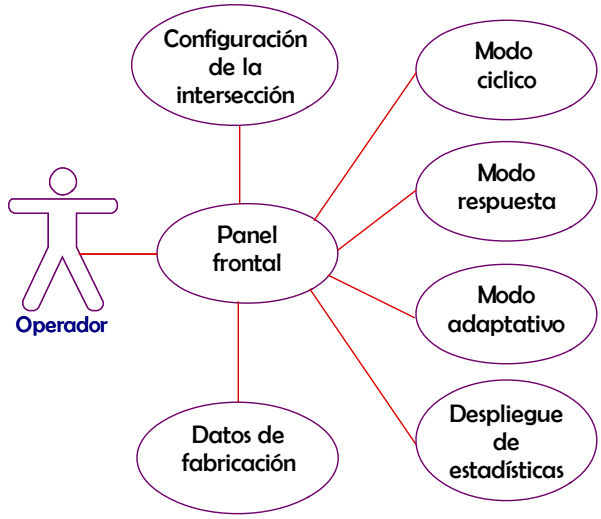


Figura 6.4. Casos de Uso *panel frontal* modelados con Rhapsody

Con el DCU se identificaron los límites del sistema y se uso como herramienta de comunicación. Es importante recalcar que no todos los casos de uso se obtienen de forma completa durante la fase dos de la metodología, por lo que SPIES propone un proceso iterativo, para ir refinando los requisitos y no correr el riesgo de construir un SE incorrecto.

Para realizar esta parte del diseño del SE es importante considerar que un Caso de Uso es una colección de requisitos operacionales que se relaciona directamente con la calidad del Sistema. Por otra parte, un Caso de Uso puede representar de 8 a 20 páginas de requisitos detallados y son ante todo requisitos funcionales que indican qué hará el sistema.

Por cuestiones de espacio, no se presentan todos los casos de uso. Solo se ejemplifican los significativos.

6.1.3. DIAGRAMA DE REQUISITOS DEL SISTEMA ROADRUNNER

UML no tiene un concepto específico llamado "requisito", una manera práctica de pensar sobre los requisitos en UML es que estos son los elementos de un modelo que especifican algún aspecto del sistema desde una perspectiva externa, en vez de interna. Una práctica efectiva basada en los casos de uso es considerar y organizar los requisitos en contexto de los DCU de un sistema, para mejorar la cohesión y comprensión.

El reto para el equipo de diseño en esta fase consistió en hallar, comunicar y registrar los requisitos. Así, con ayuda de Rhapsody se visualizaron los requisitos, la estructura y el comportamiento del sistema a través de los Diagramas UML, con esto los requisitos alcanzan un significado claro para cada uno de los miembros del equipo de desarrollo. La Figura 6.5 muestra un conjunto de requisitos generales, obtenidos con las actividades de SPIES de la página 97 (Práctica 1.2). Derivado de estas actividades se identificaron las capacidades y condiciones que el sistema debe cumplir.

El siguiente paso consistió en realizar un mapeado de los requisitos con los casos de uso (véase Figuras 6.6 y 6.7). Los casos de uso se descompusieron en casos de uso más pequeños que dicen cómo hacer las cosas; describiendo cómo se debe comportar determinado bloque del sistema.

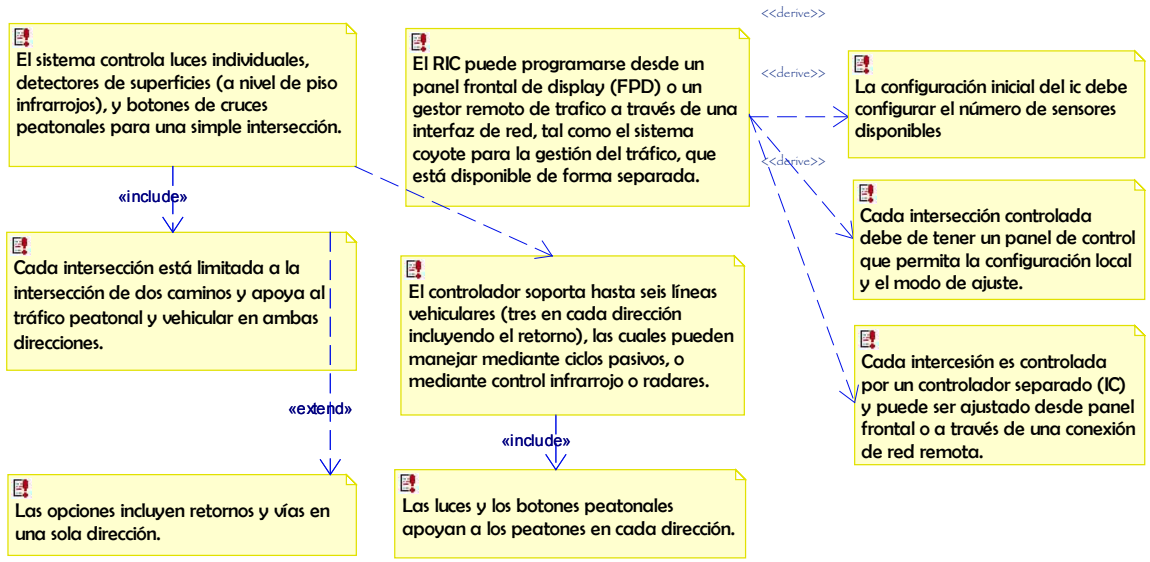


Figura 6.5. Requisitos de Roadrunner obtenidos con SPIES y modelados con Rhapsody

En la Figuras presentadas del mapeo de CU se observa el esteriotipo <<specifies>>, pues en adición a los estereotipos convencionales entre los requisitos, también se utiliza la dependencia <<specifies>> para mostrar que un caso de uso es parcialmente especificado por un requisito. Es importante recalcar que si el equipo de diseño no siguiera la metodología SPIES hubiera comenzado en cero, sin tener claros los requisitos. Con SPIES se mejora la comunicación pues se visualizan los requisitos al generar los diagramas de casos de uso y de requisitos. Así, con el desarrollo del DCU y el Diagrama de Requisitos, SPIES establece un proceso formal para comprender y obtener los requisitos de los SE. En las Figuras 6.8, 6.9, 6.10, y 6.11 se presentan el mapeo de los Casos de Uso restantes.

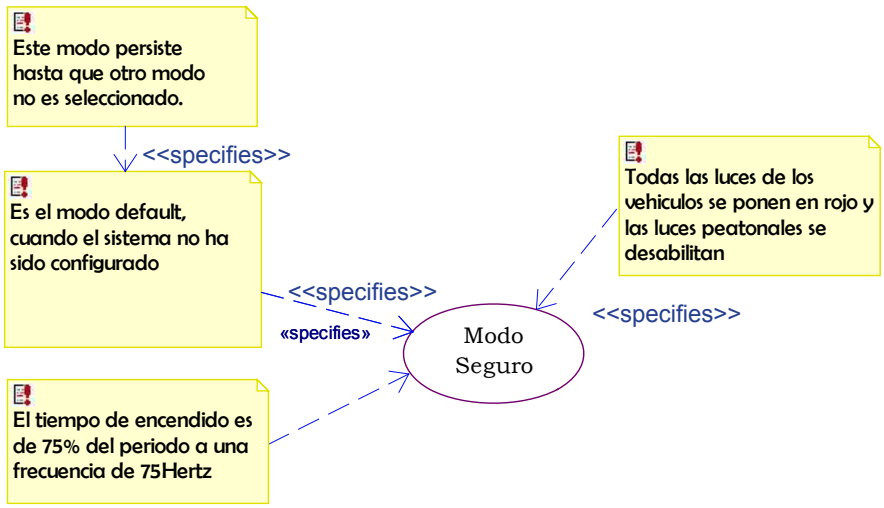


Figura 6.6. Mapeado caso de Uso de Modo Seguro

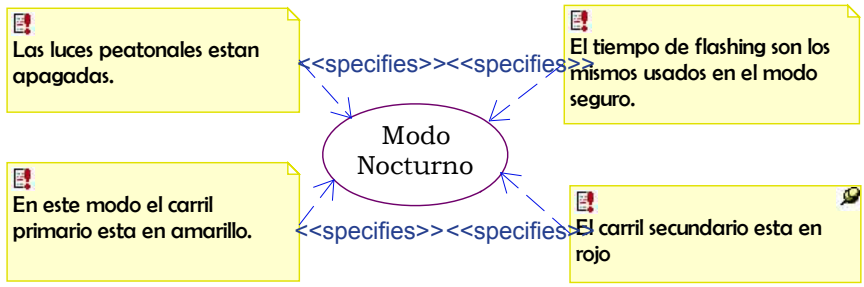


Figura 6.7. Mapeado caso de Uso de Modo Nocturno

Ahora bien, de acuerdo a la especificación del conjunto de subsistemas en los que se descompone el sistema, la ubicación de los requisitos, la funcionalidad de los subsistemas y las interfaces entre ellos, se obtiene la arquitectura del sistema. Con la arquitectura del sistema, las porciones de los mismos pueden ser manejados por los equipos de desarrollo de acuerdo a la disciplina. El objetivo de desarrollar una arquitectura del sistema consiste en identificar las desiciones estratégicas de diseño que afecten al sistema. El diseño funcional del SE *Roadrunner* cubre las funcionalidad del sistema sin considerar detalles técnicos de la implementación.

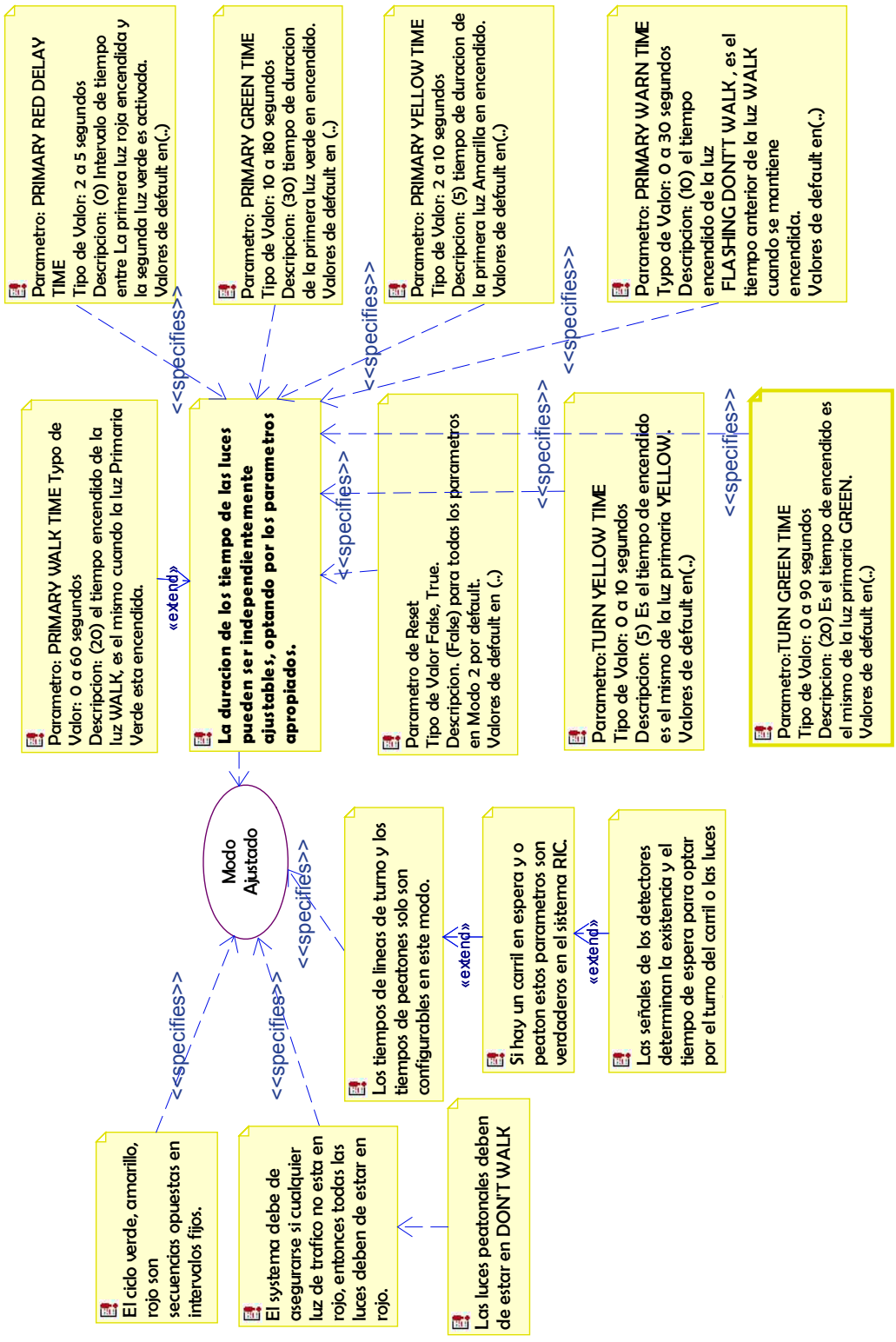


Figura 6.8. Mapeado caso de Uso de Modo Ajustado

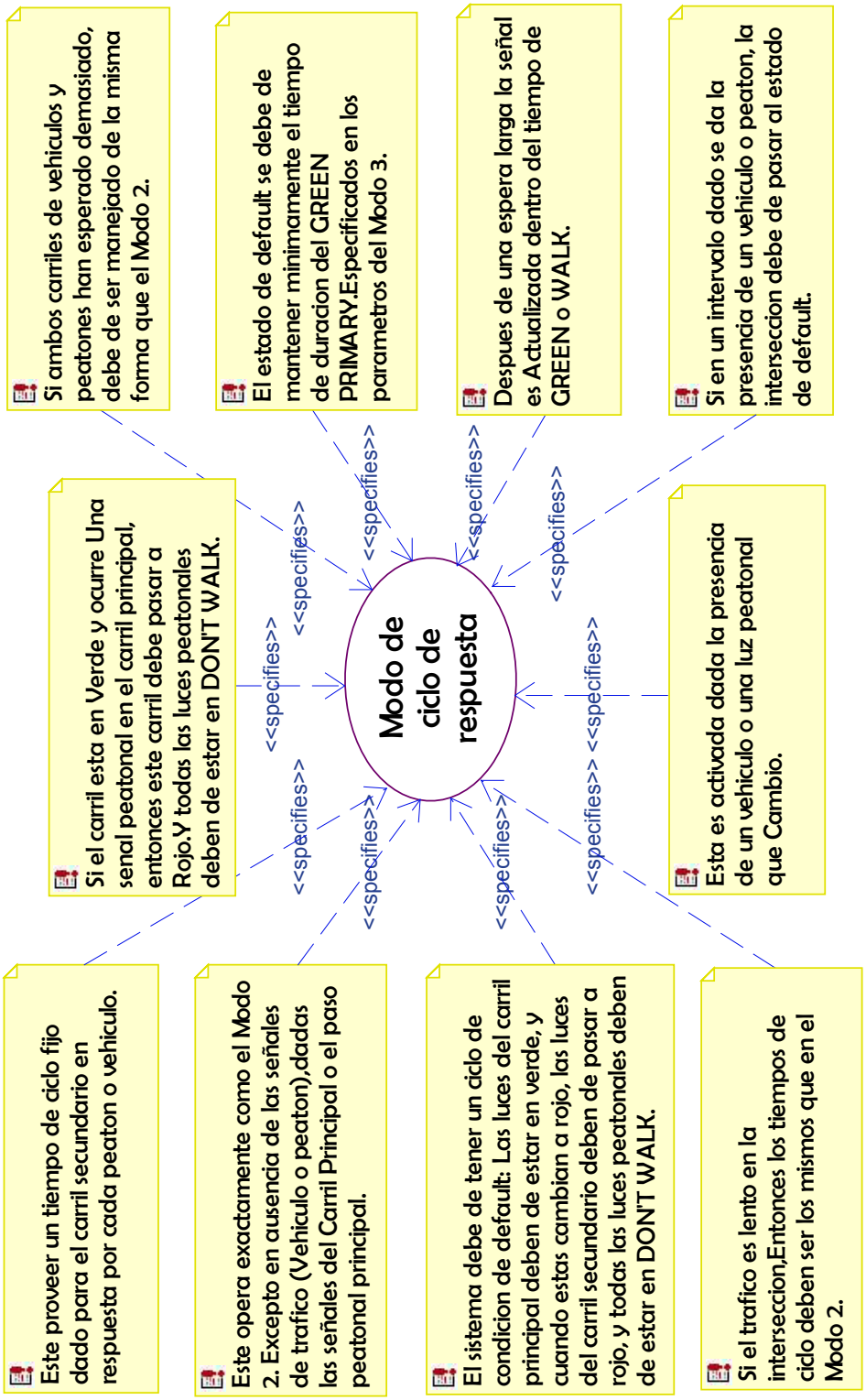


Figura 6.9. Mapeado caso de Uso de Modo Ajustado

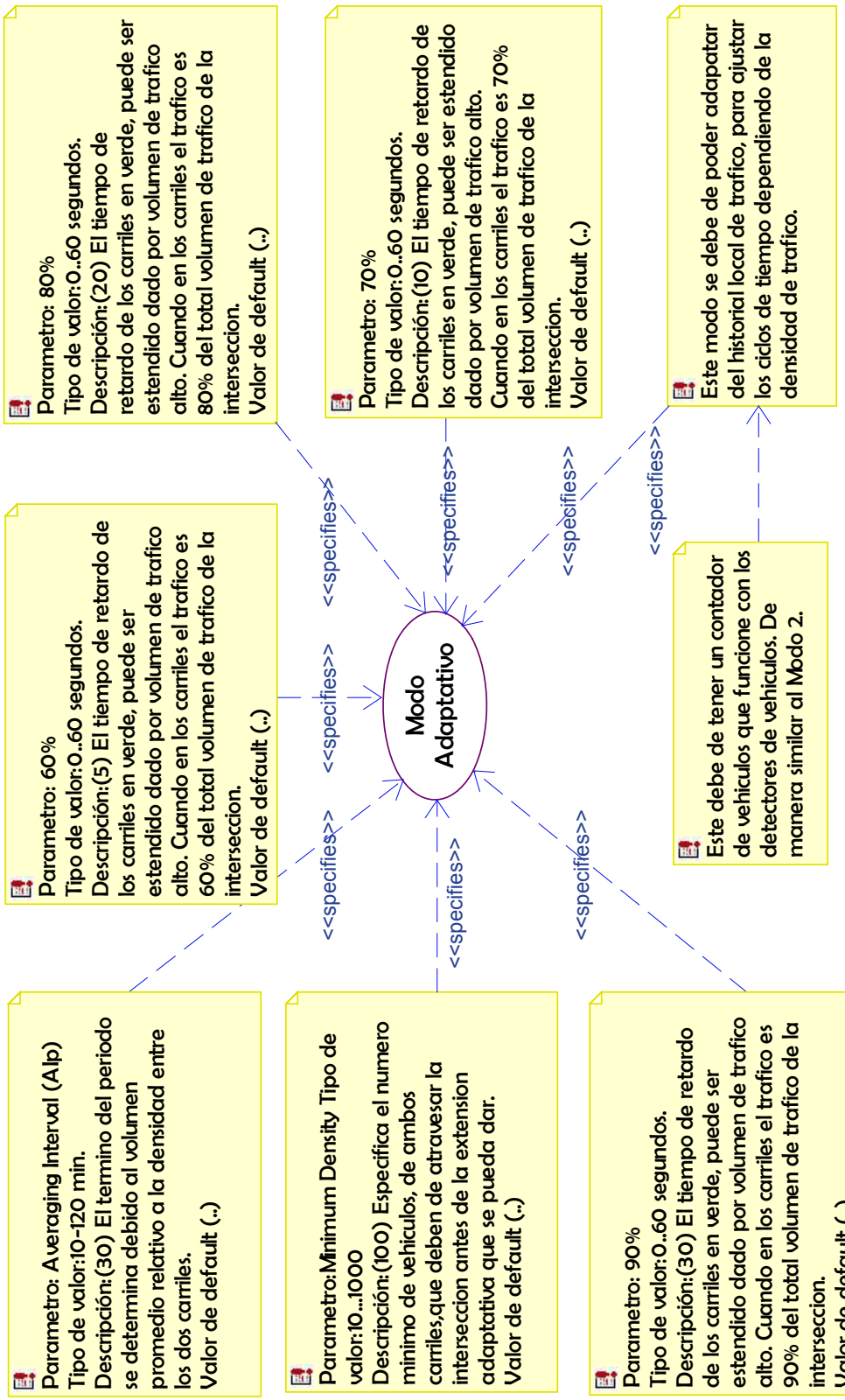


Figura 6.10. Mapeado caso de Uso de Modo Adaptativo

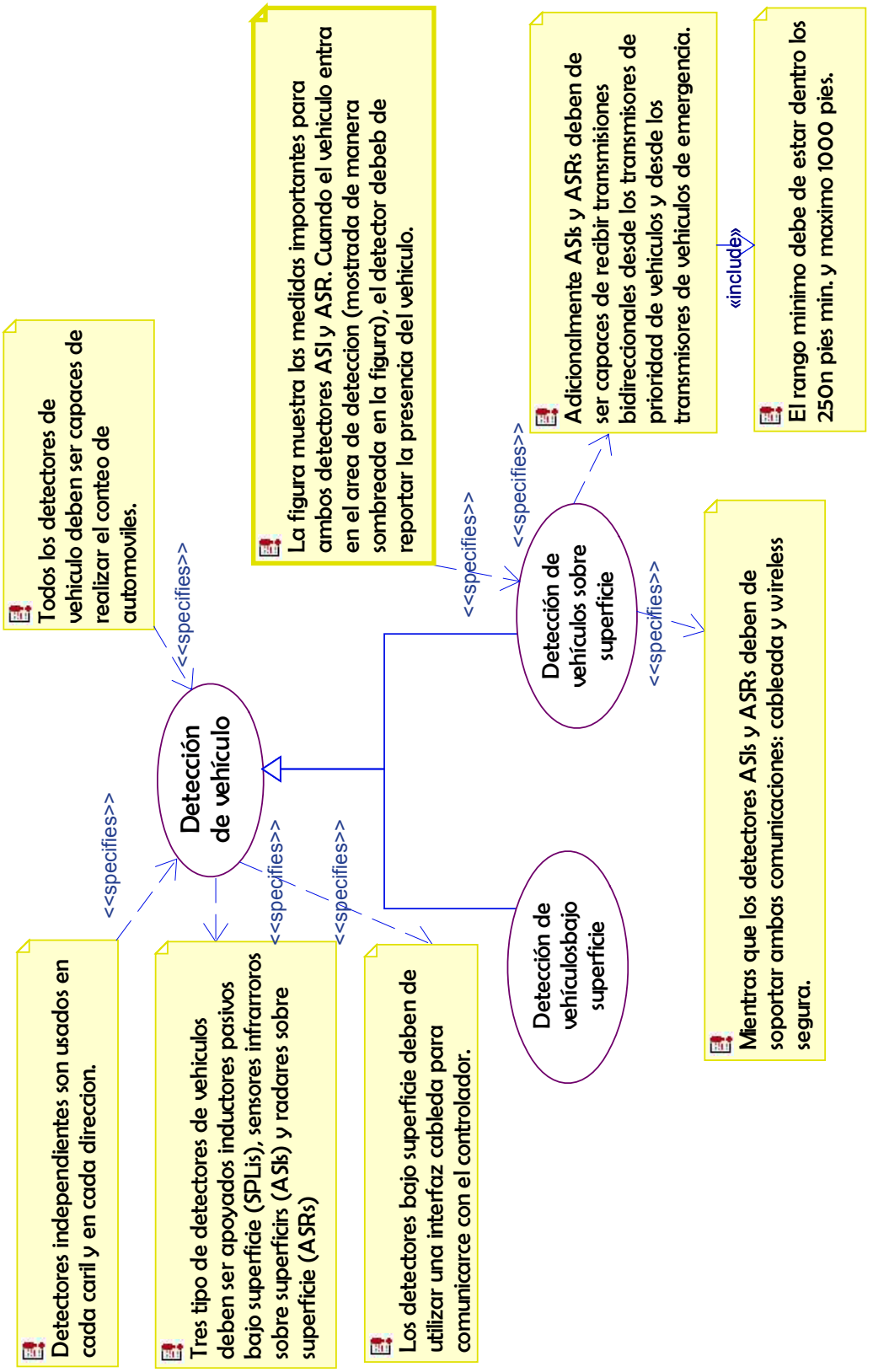


Figura 6.11. Mapeado caso de Uso Detección de vehículo

6.1.4. DISEÑO DEL SISTEMA DE ROADRUNNER

Una vez que el equipo de diseño analizó el sistema con las actividades propuestas por SPIES transformó los requisitos en modelos formales para iniciar con el *diseño* del SE. Al convertir los requisitos en modelos se aseguró la interacción de los requisitos. En esta fase el equipo de diseño fucionó la arquitectura hardware construida entorno al microcontrolador ATmega8, con la arquitectura software que involucró el lenguajes de programación C++. Además de integrar la arquitectura hardware y software para obtener la arquitectura global del SE.

6.1.5. DIAGRAMAS DE ESCTUCTURA DEL SISTEMA ROADRUNNER

Los diagramas de estructura se desprenden de los diagramas de casos de uso y definen la estructura del sistema e identifican piezas de larga escala. Para realizar los diagramas de estructuras que muestran un flujo de información entre los componentes del SE y la interfaz, se descompuso el sistema en componentes para mostrar los subsistemas y el flujo de los datos (esto es, aplicar una perspectiva de caja blanca). Las Figuras 6.10 y 6.11 presentan los diagramas de estructura generados por el equipo de desarrollo del proyecto *Roadrunner*. Para realizar este diagrama se identificaron las clases, interfaces, puertos y las relaciones. Para el diagrama de control de intercepción (véase Figura 6.12) se identificaron los siguientes elementos del diagrama:

Clases:	Puertos:	Interfaces:
• DeteccionDePeaton • DeteccionDeVehiculo • AdministracionDeTrafico • ConfiguracionDelSistema • ModoAdaptativo • ModoSeguro • ModoNocturno • ModoAjustado • ModoCicloRespuesta	• BotonPeaton • Config_Nueva • Config_Nueva_Req • Config_Sistema_OReq • Config_Sistema_RQReq • Detect_Peaton • Detect_Peaton_Req • Detect_Vehiculo • Detect_Vehiculo_Req • Mod_Adap_Port • Mod_Adap_Req • Mod_Ajus_Port • Mod_Cicl_Port • Mod_Cicl_Resp_Req • Mod_Noc_Port • Mod_Noc_Req • Mod_Seg_Req • MonitorRemoto_Req • Operador_Req • Peaton_Req	• In • Out • Config_msn_In • Config_msn_Out

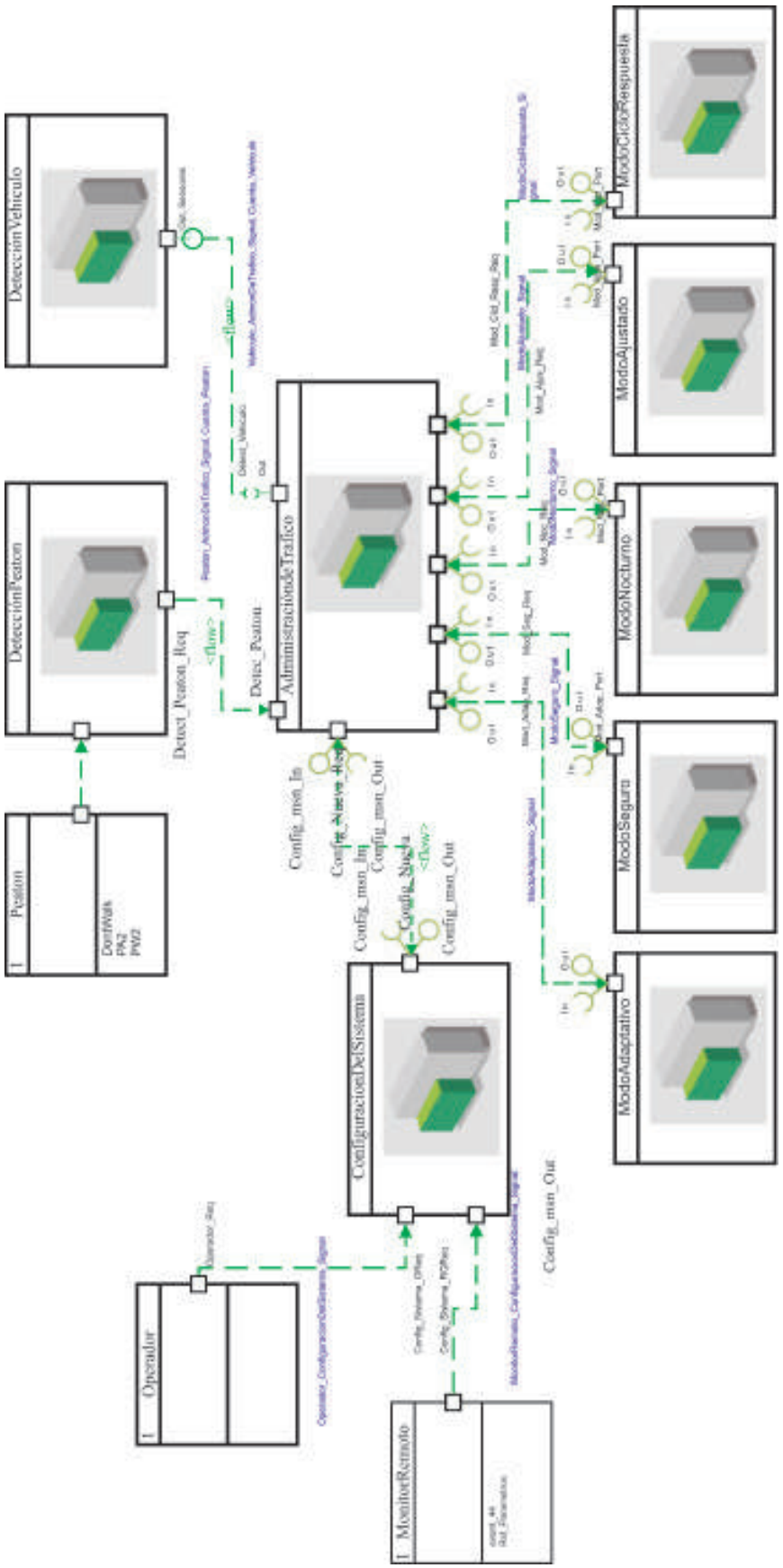


Figura 6.12. Diagrama de estructura del controlador de intersección

Para identificar las interfaces, se siguió el siguiente criterio: *"una clase realiza una interfaz si ésta proporciona un método para cada operación especificada en la interfaz, y esos métodos tienen los mismos nombres, parámetros, valores de retornos, pre condiciones y post condiciones de las operaciones correspondientes en la interfaz"*. Por otra parte, los puertos permiten capturar la arquitectura del sistema mediante la especificación de las interfaces entre los componentes del sistema, que define las relaciones entre los subsistemas. Así, el equipo identificó los puntos de conexión de las clases y les asignaron un nombre relacionado con la clase. De esta manera se modeló el sistema como una caja negra.

Para dividir la complejidad del sistema el equipo desarrolló los diagramas de estructura de la Figura 6.13 siguiendo los pasos de la Página 104 (Práctica 1.1). Los diagramas de estructura de los Modos Seguro, Modo Nocturno, Modo de Ciclo Respuesta y Modo Ajustado muestran los puertos y las interfaces y a partir de estos se puede realizar un diagrama de estructura general.

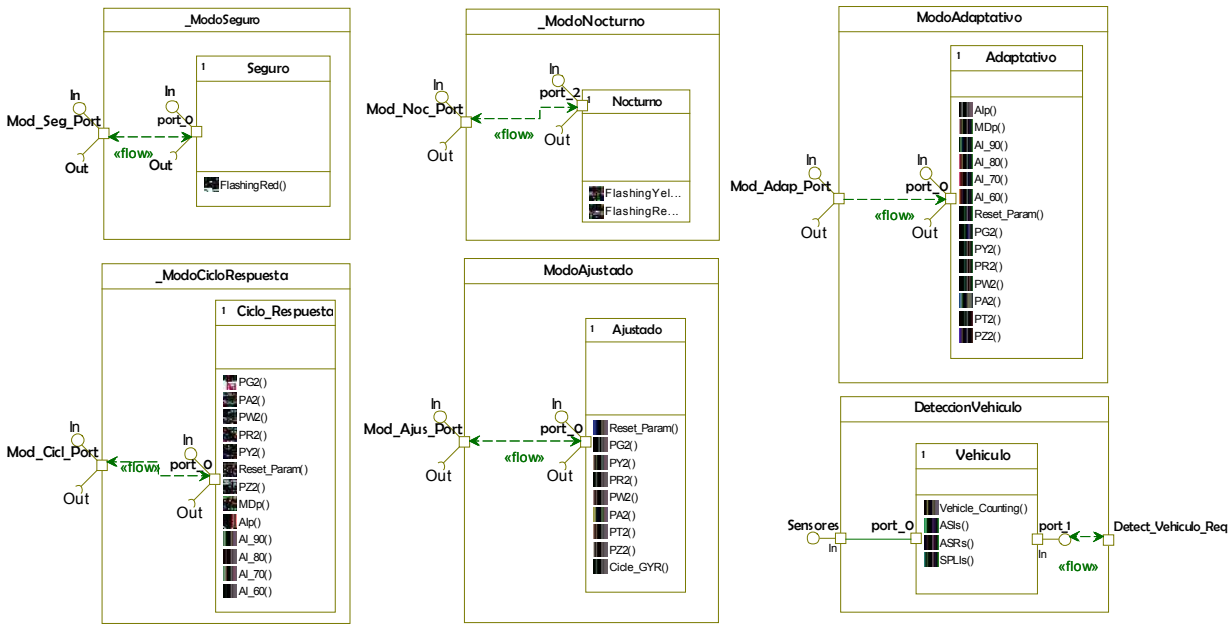


Figura 6.13. Diagrama de estructura de los modos de operación de la intersección

6.1.6. DIAGRAMAS DE SECUENCIA DEL SISTEMA ROADRUNNER

Para describir la forma en que los elementos estructurales se comunican con otros en un periodo de tiempo, se identificaron las relaciones y los mensajes requeridos y se realizaron los diagramas de secuencia. Los diagramas de secuencia se pueden crear de dos modos:

- **Modo de análisis:** se dibujan secuencias de mensaje sin agregar elementos al modelo. Esto significa que el análisis y diseño se puede realizar sin afectar el código generado.
- **Modo de diseño:** cada instancia y mensaje creado o renombrado puede verse como un elemento (clase, objeto, operación o evento para el cual se puede generar un código).

Este tipo de diagramas muestran cómo interactúan los sistemas durante un escenario de éxito de un caso de uso. A continuación se presentan los diagramas de secuencia realizados para caso de uso “Modo Ajustado”. Para crear estos diagramas de secuencia el equipo de desarrollo identificó los actores, los roles clasificadores, los mensajes y las interacciones y siguió los pasos propuestos por la metodología en la página 115 (Práctica 3.1). En las Figuras 6.14, 6.15 y 6.16 se muestran los diagrama de secuencia y por lo tanto se detallan los eventos de entrada y salida relacionados al SE, así como también los eventos que parten de los actores externos (Peatón y Operador) hacia el sistema (Modo Ajustado).

Para el diseño se permiten de 3 a 36 diagramas de secuencia por subsistema; por cuestiones de espacio sólo se presentan algunos ejemplos.

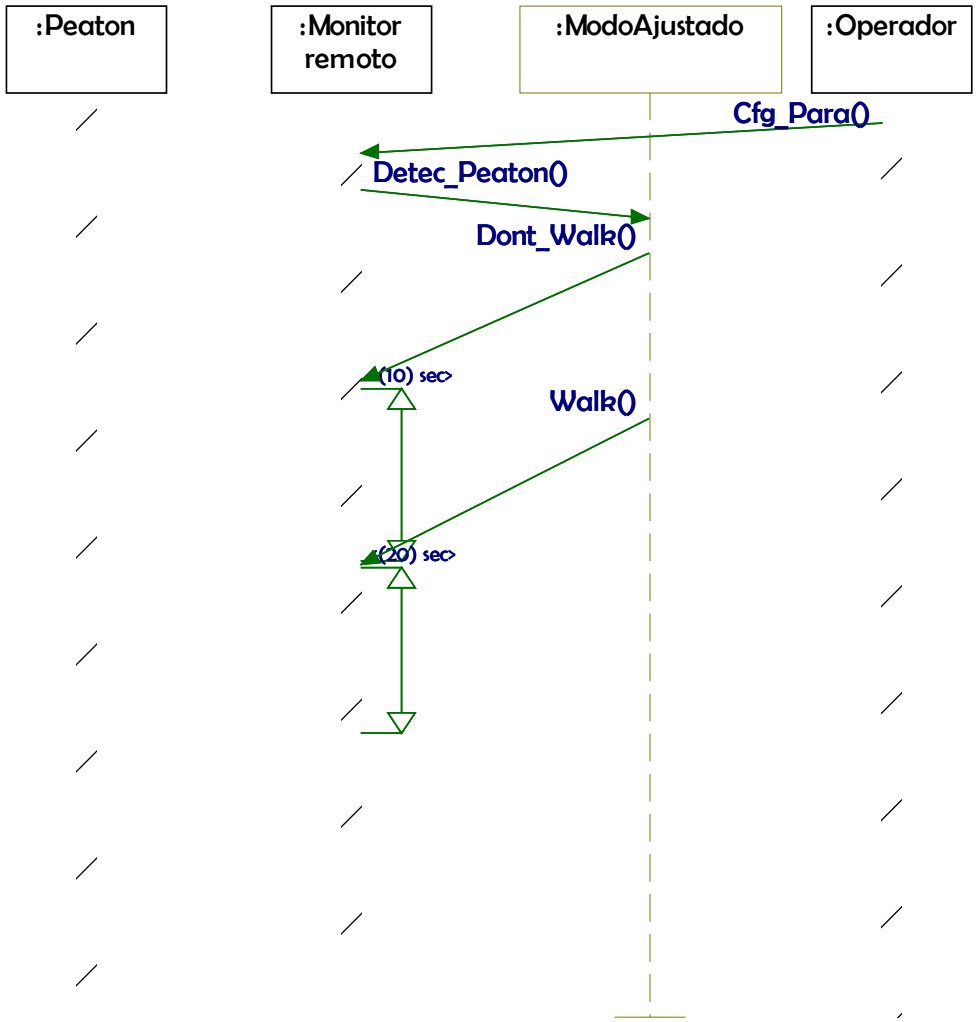


Figura 6.14. Diagrama de secuencia: un peaton se aproxima del camino B cuando el camino A esta en verde.

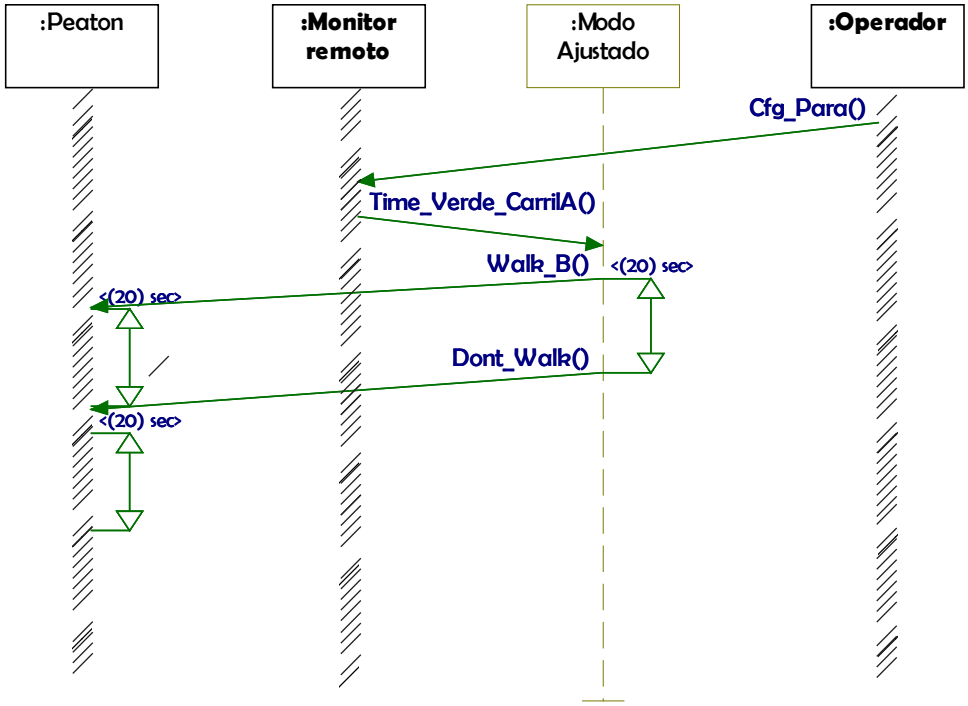


Figura 6.15. Diagrama de secuencia: un vehiculo se aproxima desde el camino B cuando el camino A esta en verde para girar

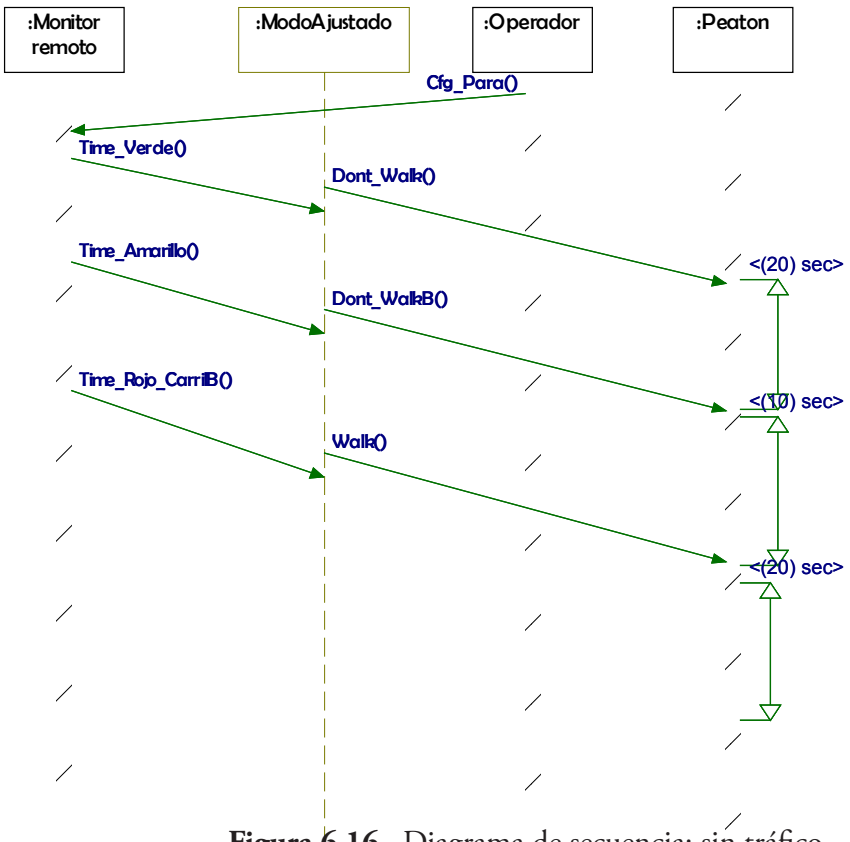


Figura 6.16. Diagrama de secuencia: sin tráfico

6.1.7. DIAGRAMAS DE MÁQUINAS DE ESTADO DEL SISTEMA ROADRUNNER

Las máquinas de estado definen el comportamiento de los actores (casos de uso o clases), los objetos y los estados a los que puede entrar sobre su línea de tiempo y los mensajes, eventos u operaciones que causan transición de un estado a otro. Estos diagramas son propuestos por la metodología para verificar el flujo funcional a nivel de diseño. Los modelos de máquina de estado que se presentan en la Figuras 6.17, 6.18, 6.19 y 6.20 modelan el comportamiento de un solo objeto, especificando la secuencia de eventos por los que el objeto atraviesa durante su tiempo de vida en respuesta a los eventos.

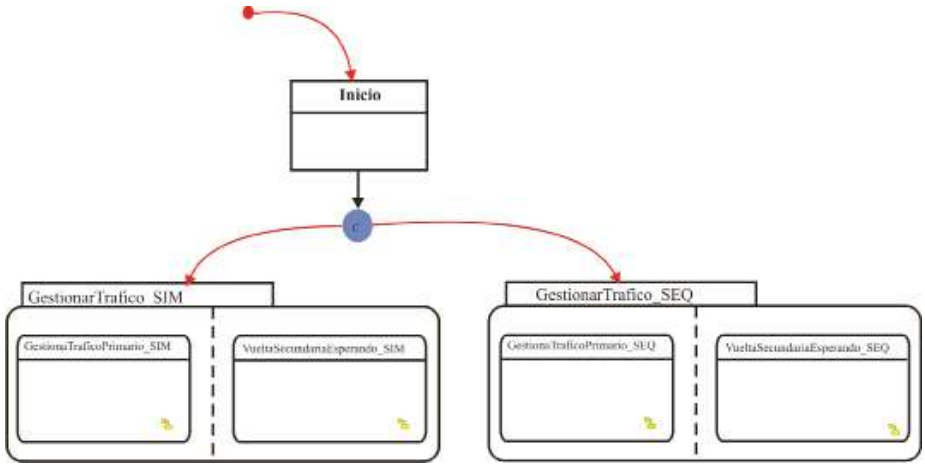


Figura 6.17. Diagrama de máquina de estados: trafico primario

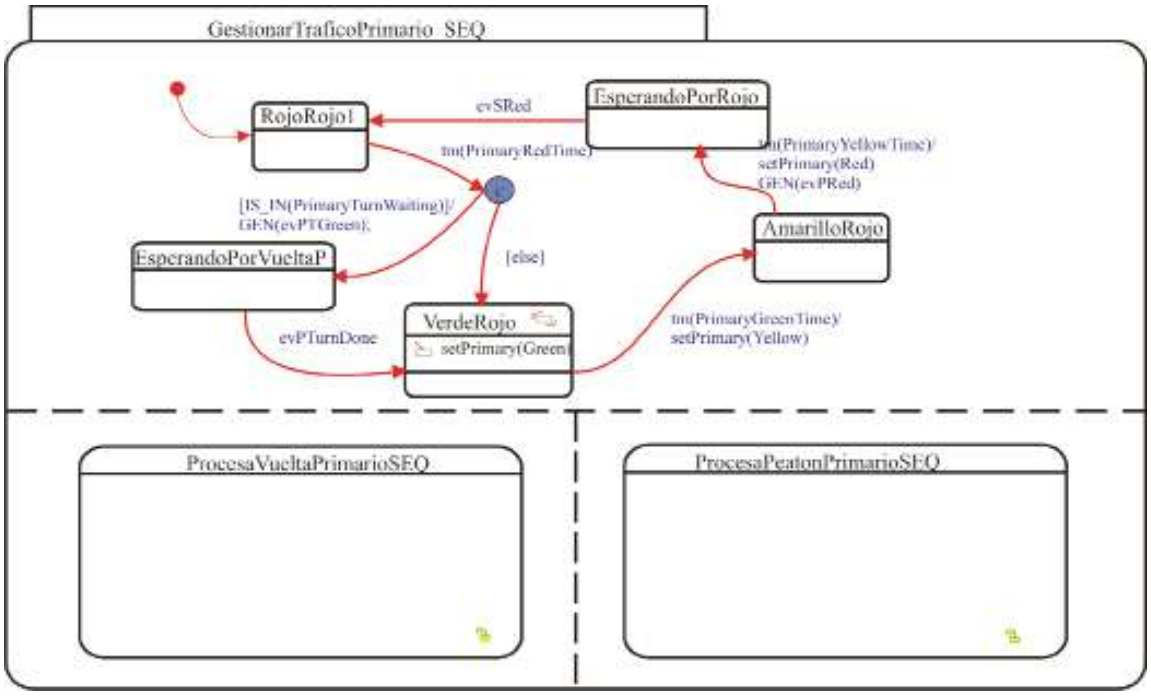


Figura 6.18. Diagrama de máquina de estados: GestionarTraficoPrimario_SEQ

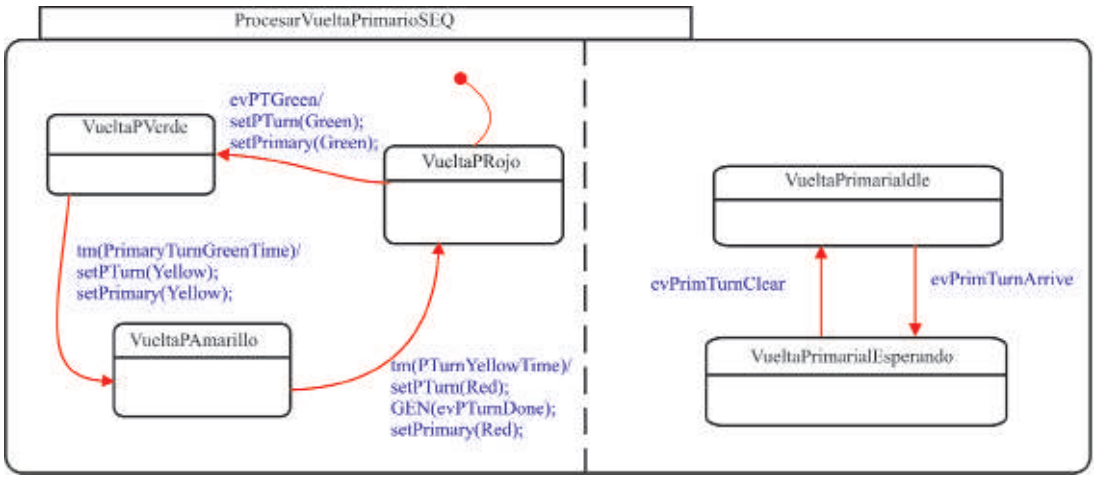


Figura 6.19. Diagrama de máquina de estados: ProcesarVueltaPrimarioSEQ

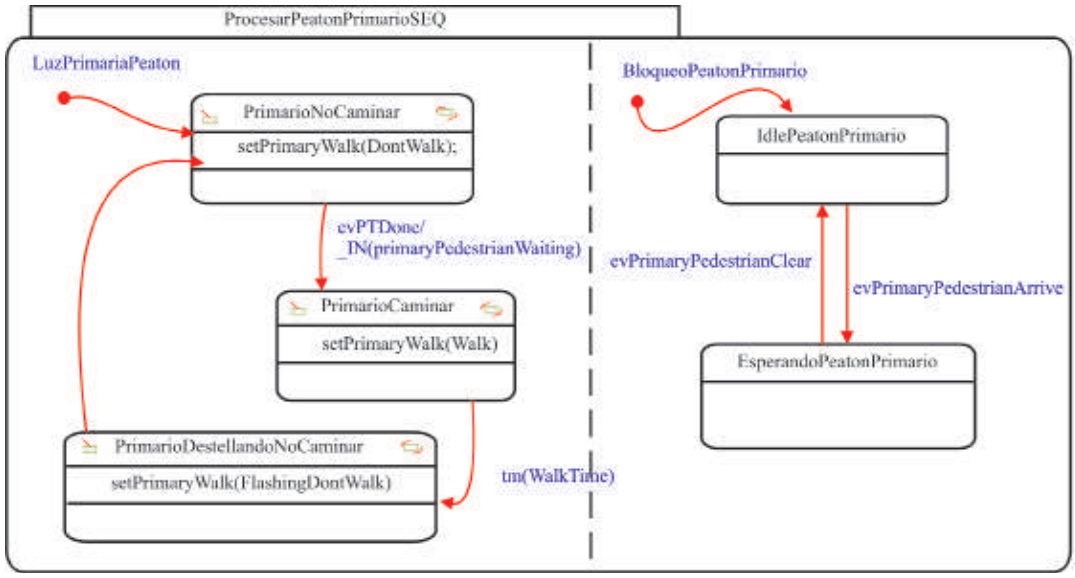


Figura 6.20. Diagrama de máquina de estados: ProcesarPeatonPrimarioSEQ

La Figura 6.21 muestra el sistema Roadrunner ya terminado. Para desarrollar el SE Roadrunner se utilizaron las áreas de proceso seleccionadas en la Figura 6.22. Es importante mencionar que SPIES considera como opcionales elaborar los artefactos y sugiere que el equipo de desarrollo elija sólo aquellos que añaden valor a su proyecto. El equipo de desarrollo debe tener presente que lo importante de un artefacto no es el documento o diagrama en sí mismo, sino el análisis y conocimiento que éste pueda registrar para evitar gastar tiempo y dinero en volver a generarlo.

Figura 6.21. Sistema prototipo de Roadrunner

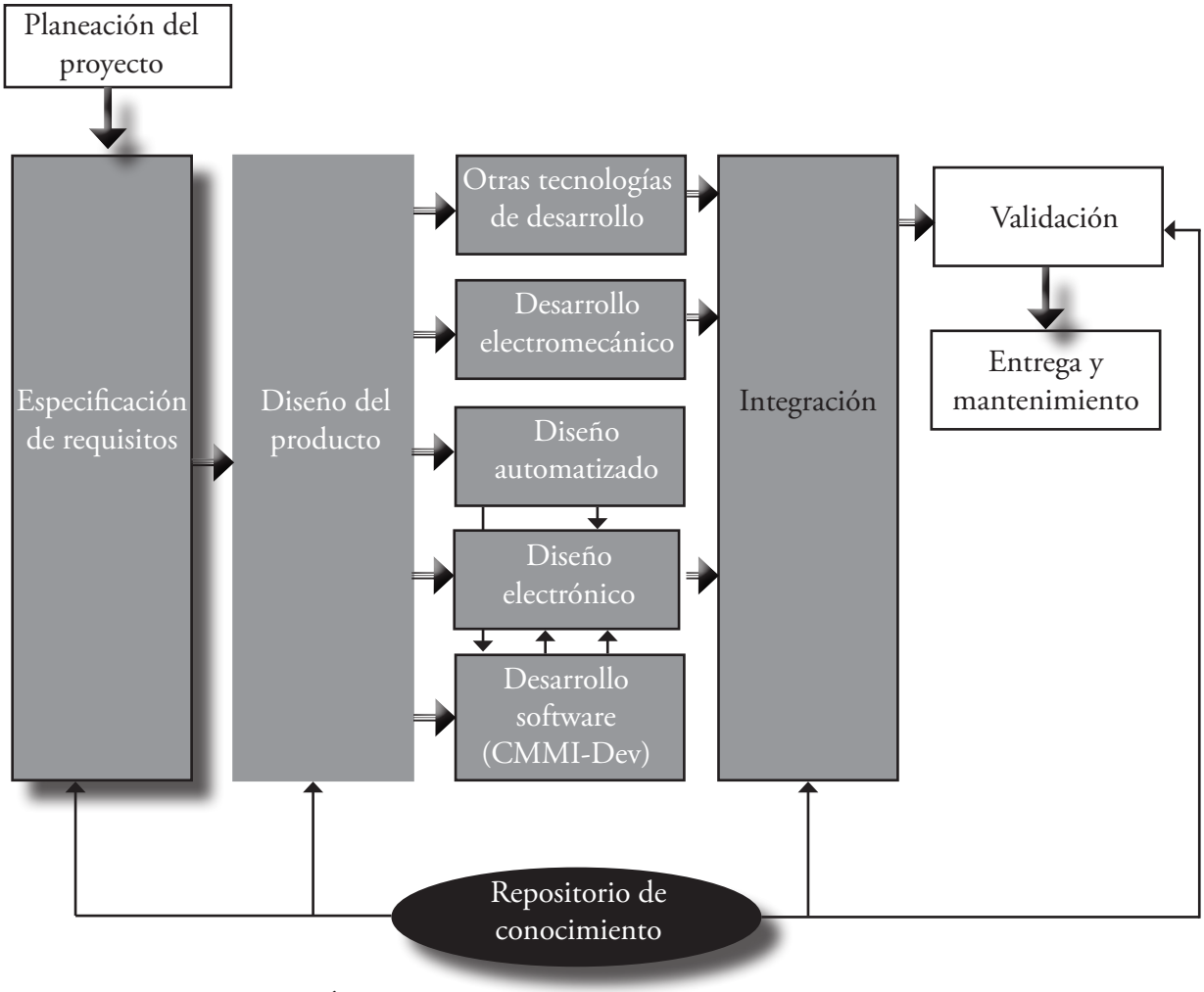


Figura 6.22. Áreas de proceso de SPIES abarcadas por el equipo de diseño de Roadrunner

6.2. SISTEMA THE COYOTE UNMANNED AIR VEHICLE SYSTEM

El Sistema de Vehículo Aéreo "Coyote" no tripulado (CUAVS) es una sistema de solución para reconocimiento de alcance medio en ambientes hostiles con capacidad de ataque limitada. Es un sistema UAV de alcance medio y larga duración que puede cargar una variedad de cargas útiles para asistir operaciones en tierra, aire y mar. CUAVS completo consiste en cuatro UAVs, planeación de misión coyote y un sistema de control (véase Figura 6.23).

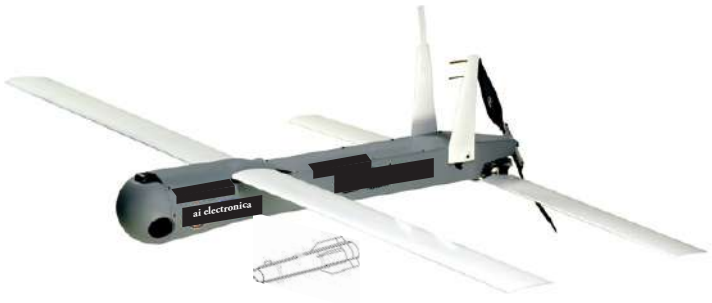


Figura 6.23. COYOTE UNMANNED AIR VEHICLE SYSTEM

6.2.1. ESPECIFICACIÓN DE REQUISITOS DEL SISTEMA *THE COYOTE UNMANNED AIR VEHICLE SYSTEM*

El Sistema CUAVS parte de las especificaciones proporcionadas en el libro *Real-Time UML Workshop for Embedded Systems* [Powel, 2006]. En el siguiente párrafo se presenta una breve descripción de las especificaciones proporcionadas.

El sistema CUAVS debe ser una solución de reconocimiento de rango medio en ambientes hostiles con capacidades de ataque limitadas. Es un sistema de rango medio, con sistemas UAV de larga resistencia que puede llevar una variedad de carga útil para apoyar en operaciones de tierra, aire y mar. Un CUAVS completo consta de cuatro vehículos aéreos no tripulados Coyote (CUAVS) y un campo de planificación de misiones Coyote y un sistema de control.

El equipo de desarrollo inició por definir las propiedades del sistema que desarrollaría. Por lo tanto, a partir de las especificaciones del sistema se realizó la fase de *especificación de requisitos*. Para esta fase el equipo de desarrollo utilizó el Artefacto ASIREQ, para presentar los requisitos de una forma consistente y entendible para todos los miembros del equipo; en este artefacto se especifican los requisitos por componente de producto, los requisitos en sí, el tipo de requisito, la fuente y la prioridad. A continuación se presenta la fase de especificación de requisitos (Artefacto ASIREQ) y los diagramas de casos de uso (Artefacto DCUF) que describen el sistema en términos de las distintas formas en que se utiliza. En cuanto al diseño se exhiben los Artefactos de los Diagramas de secuencia, de máquina de estados y de estructura.

El trabajo que se muestra a tarves de los Artefactos es producto del trabajo realizado en un tiempo no mayor a seis meses, por dos personas de la Maestria.

Artefacto ASIREQ – **Asignación de requisitos por componente de producto**

Fecha: _____ **Ciclo:** _____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López

Descripción general	
Perspectiva del producto	
Es un sistema de reconocimiento de alcance medio en ambientes hostiles con una capacidad de ataque limitado, contiene 4 vehículos aéreos no tripulados y un sistema de control en tierra.	
Funcionalidad del sistema	
1.	Reconocimiento de objetivos en entornos hostiles y ataque limitado.
2.	Búsqueda y rastreo de objetivos.
3.	El UAV puede volar en cualquier tipo de clima.
4.	El UAV puede llevar varios tipos de carga.
5.	El UAV puede ser controlado desde una estación en tierra (CMPCS).
6.	El CMPCS puede controlar hasta 4 UAV's.
7.	La telemetría se deberá transmitir a través de un enlace de datos de vigilancia.
8.	Diferentes modos de operación como: modo de apoyo operacional, remoto, alta fidelidad a distancia.
Características de los usuarios	
Tipo de usuario: Operador remoto	
Formación: Piloto aviador	
Habilidades:	
Restricciones	
1.	El UAV no debe ser mayor a 30 pies de largo por 10 pies de envergadura.
2.	El peso total no puede sobrepasar los 2100 lb.
3.	El UAV no puede llevar más de 4 misiles Hellfire.
4.	El UAV no puede llevar más de 450 lb. de carga.
5.	El combustible está limitado a 24 horas de vuelo.
6.	El máximo alcance para la navegación remota es de 400 millas nauticas.
Suposiciones y dependencias	
1.	Exceder el limite de peso en la carga del UAV.
2.	La distancia de alcance del control remorto sea mayor a las 400 millas nauticas.
3.	
Evolución previsible del sistema	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos				
Número de requisito:	Req 1.1			
Nombre del requisito:	El *Coyote Unmanned Air Vehicle System* (CUAVS) es un sistema de reconocimiento de alcance medio en ambientes hostiles con una capacidad de ataque limitado.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 1.2			
Nombre del requisito:	El CUAVS es un sistema UAV de alcance medio de gran resistencia que puede llevar una gran variedad de cargas para asistir operaciones en tierra, aire y mar.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 1.3			
Nombre del requisito:	Un CUAVS completo consiste en 4 vehículos aéreos no tripulados (CUAV's) y un sistema de control en tierra (*Coyote Mission Planning and Control CMPCS*).			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.1			
Nombre del requisito:	El UAV Coyote está destinado a ser un UAV multipropósito y reutilizable con capacidad para múltiples misiones.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.2			
Nombre del requisito:	EL UAV coyote está destinado para operar a altitudes hasta de 30, 000 pies con velocidades de avance de hasta 100 nudos (crucero) y 150 nudos (de escape), y llevar una carga de hasta 450 libras y con una duración superior a las 24 hrs.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos				
Número de requisito:	Req 2.3			
Nombre del requisito:	El Coyote está destinado a volar sin obstáculos en ambientes de baja visibilidad mientras lleva cargas de reconocimiento o ataque.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.4			
Nombre del requisito:	Mientras el CUAV es controlado desde una estación en tierra (CMPCS) es capaz de volar planes complejos de vuelo con objetivos operacionales específicos de una zona de búsqueda sistemática, buscar en tierra y orbitas de vigilancia para objetivos.			
Tipo:	Operacional [x]	Funcional	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.5			
Nombre del requisito:	Junto con el control tripulado desde tierra, el coyote proporciona 24 hrs. de vuelo sostenido con visualización en tiempo real, telemetría por infrarrojo o por radar, con pre-procesamiento de reconocimiento de objetivos.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.6			
Nombre del requisito:	Las comunicaciones son resistentes a la interferencia, aunque no necesitan ser anti interferencia en un ambiente con un alto ECM.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.7			
Nombre del requisito:	Los comandos de control deben ser encriptados mientras que los datos de telemetría se pueden comprimir pero sin protección.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 2.8			
Nombre del requisito:	Las tasas de telemetría para telemetría visual soportan 30 fps a una resolución de 640X400.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.9			
Nombre del requisito:	El rango de vuelo es completamente compatible con el rango de línea de visión (LOS) pero desde que el coyote tiene también la habilidad de ser pasado entre los diferentes CMPCS's, su alcance es considerablemente mayor que LOS.			
Tipo:	Operacional [x]	Funcional	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.10			
Nombre del requisito:	Para navegación, el coyote tiene a bordo un sistema GPS y también puede ser controlado directamente desde la estación de tierra.			
Tipo:	Operacional [x]	Funcional	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.11			
Nombre del requisito:	A diferencia de muchos UAVs más pequeños, el coyote no requiere de un despegue especializado ni vehículos de recuperación.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.12			
Nombre del requisito:	El UAV puede utilizar una pista corta para despegue o aterrizaje automático o a control remoto.			
Tipo:	Operacional [x]	Funcional	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado	Baja/Opcional [x]	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 2.13			
Nombre del requisito:	El Coyote realiza cualquiera de los patrones de búsqueda sistemática de un objetivo que cumpla con las especificaciones (como un tanque, barco, caravana, pelotón, soldado, plataforma de lanzamiento de misiles, edificios) al tiempo que notifica al CMPCS.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.14			
Nombre del requisito:	El vehículo aéreo no tripulado (UAV) no deberá ser de más de 30 pies de largo con una envergadura de no más de 10 pies.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.15			
Nombre del requisito:	El UAV deberá ser capaz de despegar y aterrizar usando una pista de no más de 1500 pies por 50 pies, tanto de forma automática como vía remota de navegación.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.16			
Nombre del requisito:	Cuando esté completamente cargado (450 libras de carga), el sistema deberá ser capaz de mantener por lo menos 24 horas de vuelo.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.17			
Nombre del requisito:	El peso bruto del CUAV no podrá superar 2,100 libras.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 2.18			
Nombre del requisito:	El CUAV será capaz de volar de manera confiable en escasa visibilidad e inclemencias del tiempo.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.19			
Nombre del requisito:	El CUAV se puede volar sin pérdida de fiabilidad con vientos cruzados de 15 nudos, vientos de frente de 30 nudos, y en la lluvia, el aguanieve, granizo, nieve y en condiciones gélidas.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.20			
Nombre del requisito:	El CUAV tendrá un alcance de hasta 400 millas náuticas sin peso o con no más de 1,200 libras (2,100 libras cargado).			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.21			
Nombre del requisito:	Tiene un tope de 30,000 pies y una capacidad de combustible de 100 galones (665 libras).			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.22			
Nombre del requisito:	El CUAV tendrá un despliegue de paracaídas para la recuperación de emergencia.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 2.23			
Nombre del requisito:	El CUAV es capaz de vuelo totalmente automatizado, pero es de imaginarse que estará normalmente bajo el control de navegación de la CMPCS. Los siguientes modos de vuelo son apoyados: Totalmente automatizado , Operacional, Mando a distancia (CUAV mantiene postura estable), Alta fidelidad a distancia (el piloto remoto mantiene la postura estable)			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.24			
Nombre del requisito:	En el modo totalmente automático, el CUAV puede despegar, volar una misión preprogramada y regresar.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.25			
Nombre del requisito:	Si el CUAV experimenta una pérdida inesperada de las comunicaciones con su CMPCS por más de 60 minutos, este puede abortar la misión y realizar un regreso y aterrizaje automáticos. Esto se hace para disminuir la incidencia de pérdidas de UAV debido al ECM y fallas en las comunicaciones.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.26			
Nombre del requisito:	Si no es posible aterrizar de forma automática, el CUAV desplegará su paracaídas de emergencia.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 2.27			
Nombre del requisito:	La navegación automática se llevará a cabo utilizando un sistema de navegación por inercia o un sistema GPS, o una combinación de los dos.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.28			
Nombre del requisito:	El CUAV informará su posición a lo largo del enlace de datos de control por lo menos cada 3 segundos; estos datos incluirán latitud, longitud, y altitud.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.29			
Nombre del requisito:	El CUAV será visible cuando vuele en ambientes con menos de tres satélites GPS, tendrá también a bordo un altímetro.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 2.30			
Nombre del requisito:	Más allá de los modos de vuelo, el CUAV deberá estar diseñado para parámetros altamente flexibles de una misión. Los modos de misiones normales incluyen: Reconocimiento preplaneado, Reconocimiento a distancia controlado, Búsqueda de Área, Búsqueda de Ruta, Objetivo de punto de orbita, Ataque			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 2.31			
Nombre del requisito:	Una misión puede consistir en cualquier número de presentaciones secuenciales, cada una de ellas operando en un modo de misión diferente, dependiendo de la carga actual.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 3.1			
Nombre del requisito:	CMPCS móviles con capacidad para controlar hasta 4 UAVs con una estación de control tripulada por UAV que cabe en un pequeño remolque.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.2			
Nombre del requisito:	Cada estación de control consta de 2 subestaciones tripuladas, una para el control del CUAV y otra para el monitoreo y control de las cargas. Si se desea, ambas funciones se pueden realizar en una sola subestación de control.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.3			
Nombre del requisito:	El control de la aeronave está integrado por la transferencia de comandos de navegación que pueden ser simples (fijar altitud, velocidad, dirección), operacionales (fijar coordenadas de vuelo, puntos de órbita, ejecutar un patrón de búsqueda, etc.), planeados (subir varios segmentos del plan de vuelo) o a control remoto con una interface de joystick.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.4			
Nombre del requisito:	La mecánica de vuelo podrá ser gestionada por la propia aeronave, pero esto se pude deshabilitar para el vuelo a control remoto.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.5			
Nombre del requisito:	El CMPCS se encuentra en un remolque de 30 × 8 × 8 de triple eje que contiene las estaciones para el piloto y las operaciones de carga, planificación de la misión, la explotación de datos, comunicaciones y la visión SAR.			
Tipo:	Operacional	Funcional	Diseño	Restricción [x]
Fuente del requisito:	Especificación del problema B			

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 3.6			
Nombre del requisito:	El CMPCS se conecta a múltiples antenas direccionales para la comunicación con los CUAVs, todos los datos de la misión son grabados en el CMPCS ya que el CUAV no tiene la capacidad de grabación a bordo.			
Tipo:	Operacional [x]	Funcional	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.7			
Nombre del requisito:	El CMPCS tiene un UPS que puede trabajar a carga completa hasta 4 horas, además de utilizar energía comercial y generadores de energía.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.8			
Nombre del requisito:	Un solo CMPCS puede controlar hasta 4 CUAV's en vuelo, con una estación por cada CUAV.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.9			
Nombre del requisito:	Cada estación de control CUAV proporciona operaciones piloto y de carga con subestaciones de control independientes, aunque ambas funciones se pueden realizar en una sola subestación para el uso de baja vigilancia.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.10			
Nombre del requisito:	Para las cargas de reconocimiento, el CMPCS proporcionará un mejor reconocimiento de objetivos automatizado (ATR) con capacidad para todo tipo de vigilancia (óptica, infrarroja y radar).			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 3.11			
Nombre del requisito:	Mientras que el CUAV tiene una capacidad rudimentaria, el CMPCS proporciona mucho más apoyo para la identificación rápida de objetivos de alto valor en el campo de batalla.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.12			
Nombre del requisito:	Esta capacidad está específicamente diseñada para identificar objetivos móviles y objetivos en un tiempo limitado, que solo están expuestos por breves periodos de tiempo antes de volver a ocultarse.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 3.13			
Nombre del requisito:	Se espera que el sistema proporcione un alto rechazo al (clutter) y una tasa de error baja de falsos positivos.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.14			
Nombre del requisito:	El ATR deberá ser capaz de identificar y rastrear hasta 20 objetivos dentro del área de vigilancia con identificación y evaluación de probabilidad para cada uno.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.15			
Nombre del requisito:	Para las cargas de reconocimiento, el CMPCS proporcionará un mejor reconocimiento de objetivos automatizado (ATR) con capacidad para todo tipo de vigilancia (óptica, infrarroja y radar).			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System

Requisitos específicos

Número de requisito:	Req 3.16			
Nombre del requisito:	La vista del campo de batalla se puede trasmitir a través de enlaces a personal de mando remoto para una evaluación táctica y estratégica.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 3.17			
Nombre del requisito:	Las comunicaciones incluirán radio, celulares y teléfonos fijos además de comunicación por radio con la CUAV.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

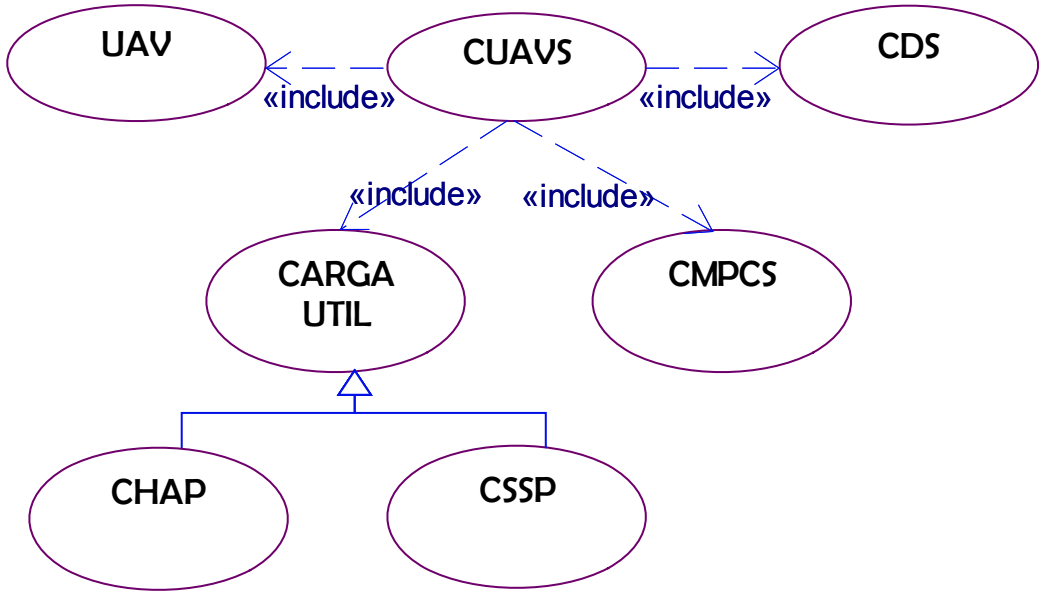
Número de requisito:	Req 4.1			
Nombre del requisito:	El coyote UAV deberá ser configurado fácilmente para controlar varias cargas de vigilancia o reconocimiento (video, FLIR (forward looking infrared, radar), cargas de contramedida (ECCM) y de ataque (misiles Hellfire).			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial [x]	Media/Deseado	Baja/Opcional	

Número de requisito:	Req 4.2			
Nombre del requisito:	El coyote se puede configurar hasta con 450 lb de carga y dentro de esta limitante puede contener una mezcla de cargas de vigilancia, ECM/ CCME y de ataque.			
Tipo:	Operacional	Funcional	Diseño [x]	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

Número de requisito:	Req 4.3			
Nombre del requisito:	Las cargas son controladas principalmente desde tierra, pero algunas funciones coma la búsqueda de objetivos se pueden asignar a la propia UAV.			
Tipo:	Operacional	Funcional [x]	Diseño	Restricción
Fuente del requisito:	Especificación del problema B			
Prioridad:	Alta/Esencial	Media/Deseado [x]	Baja/Opcional	

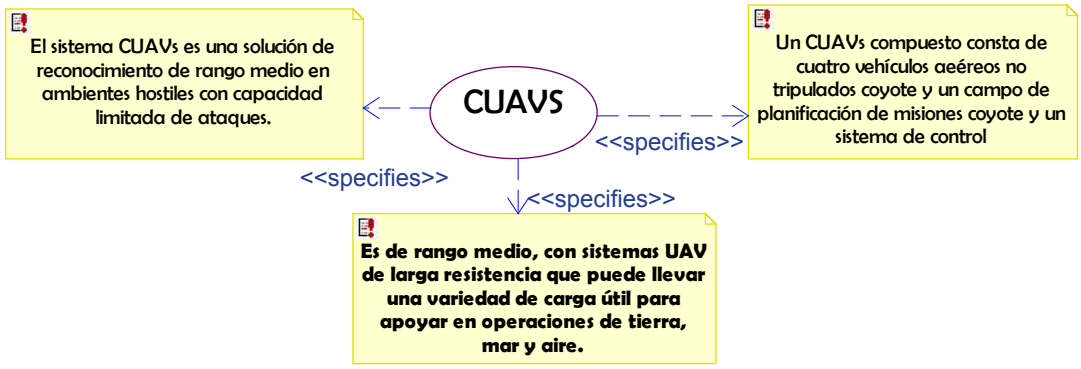
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



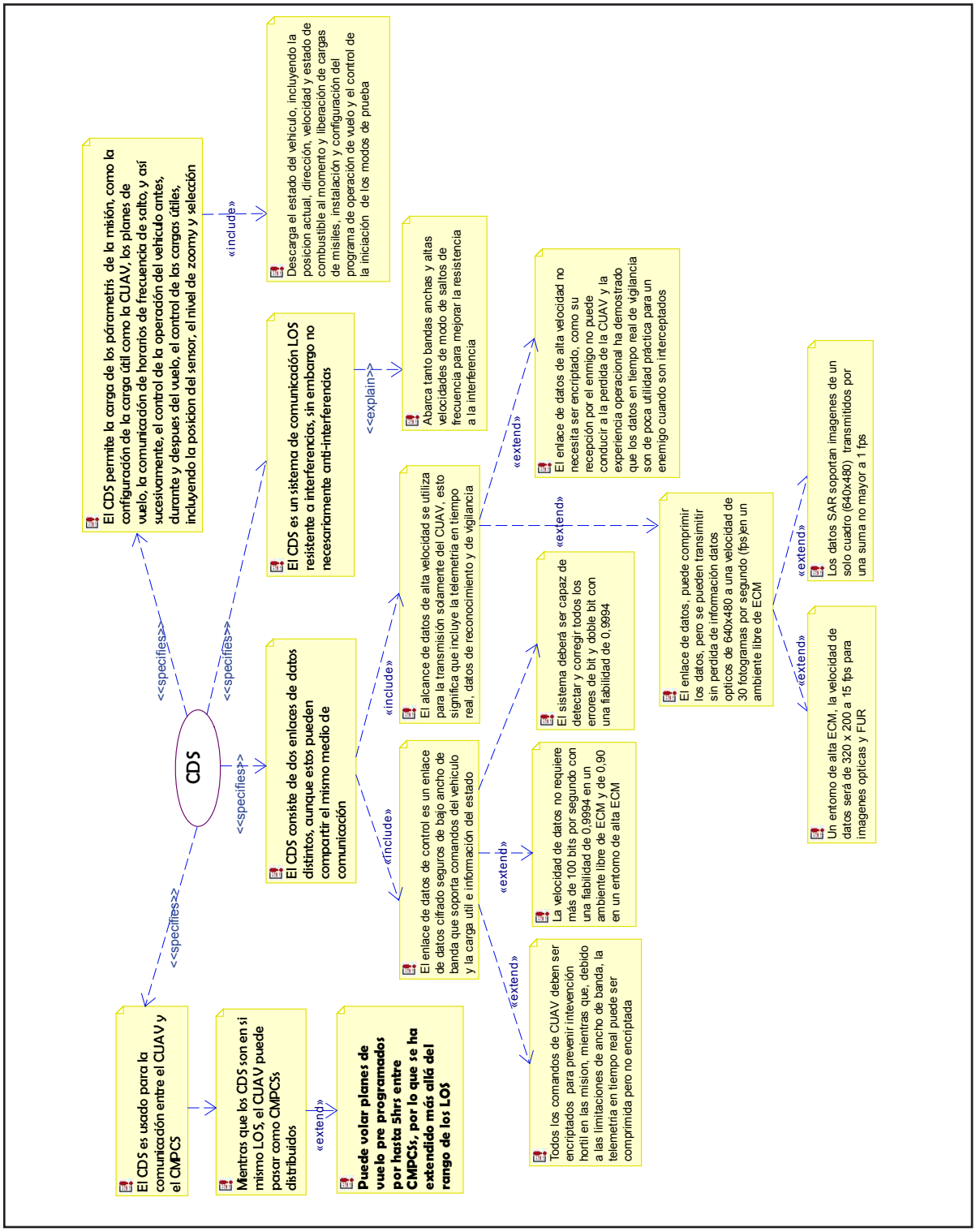
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



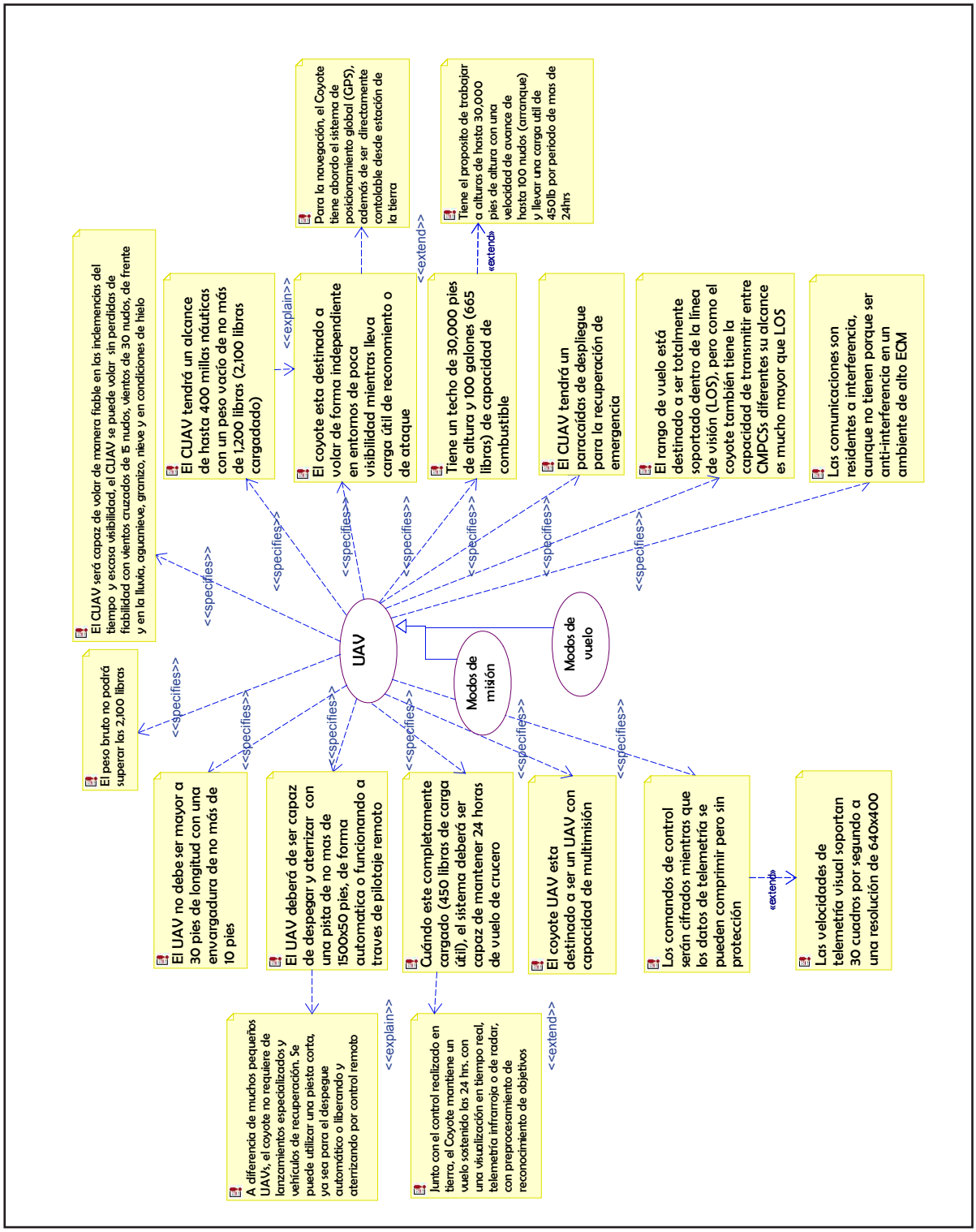
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



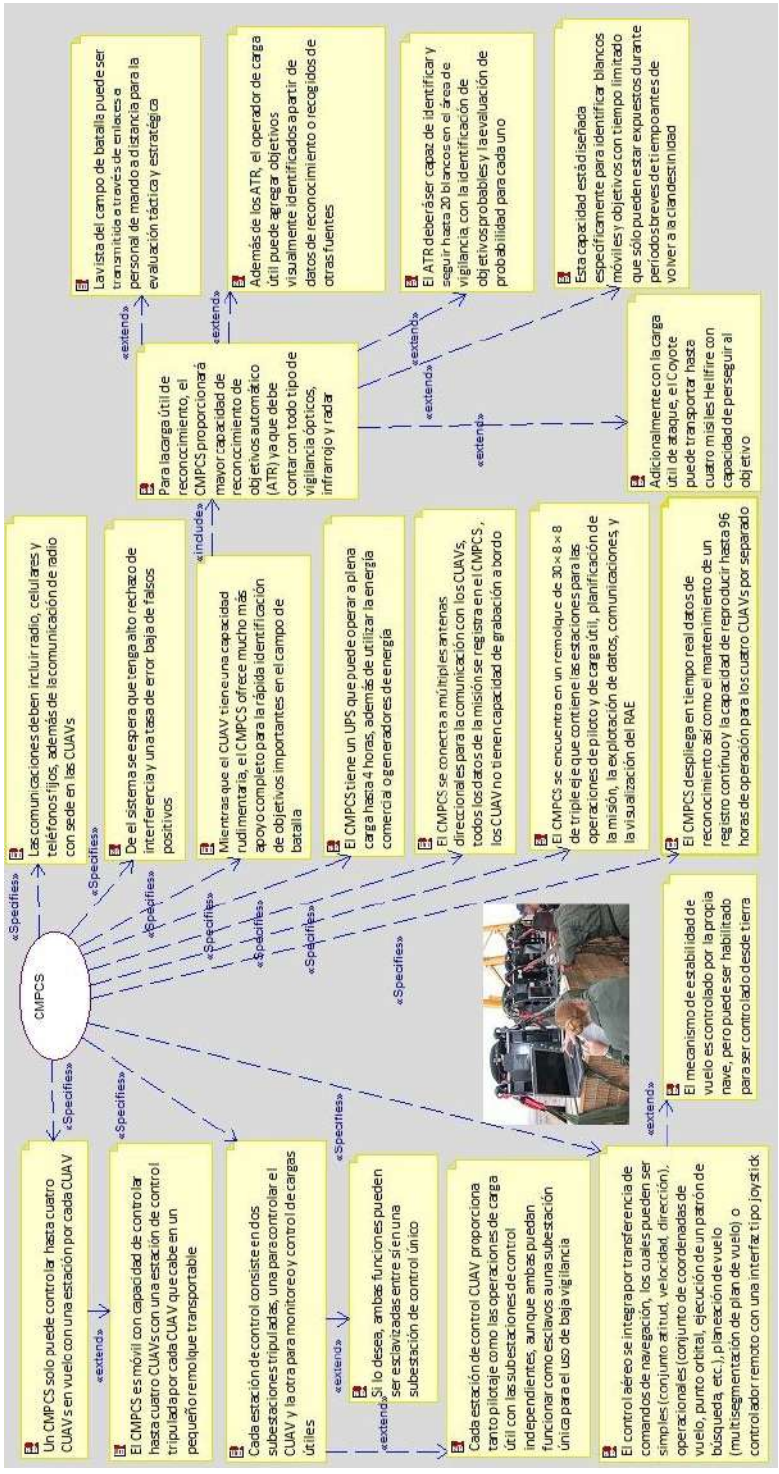
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



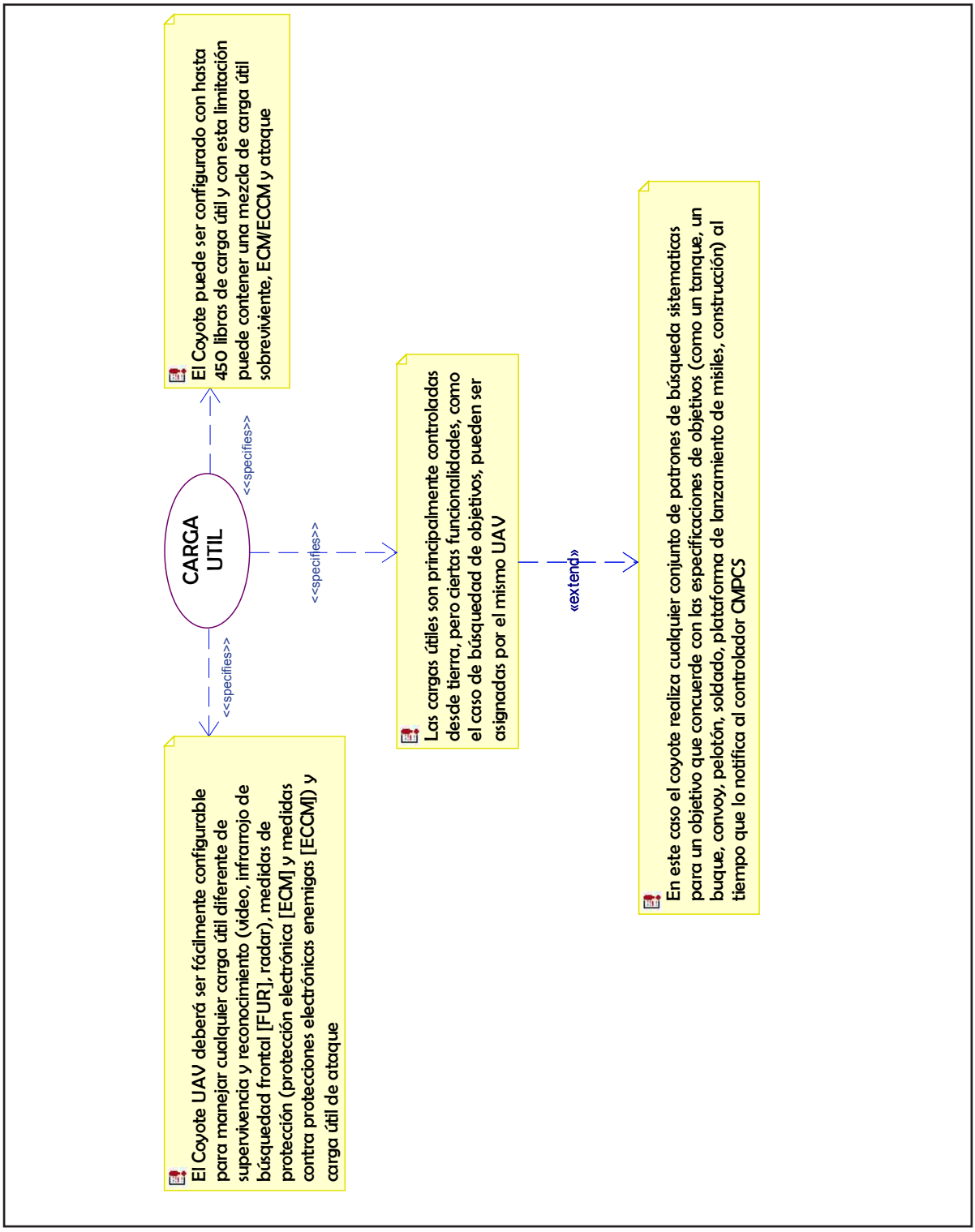
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:_____ Ciclo:_____**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



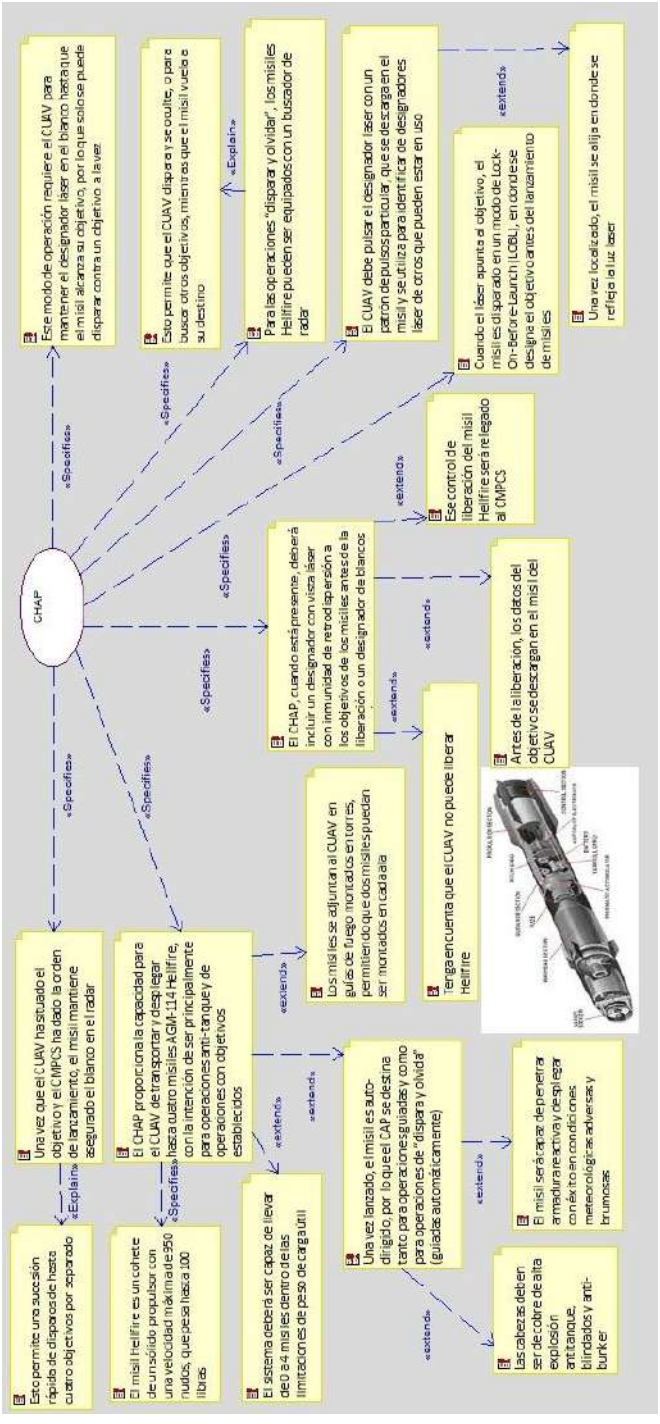
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



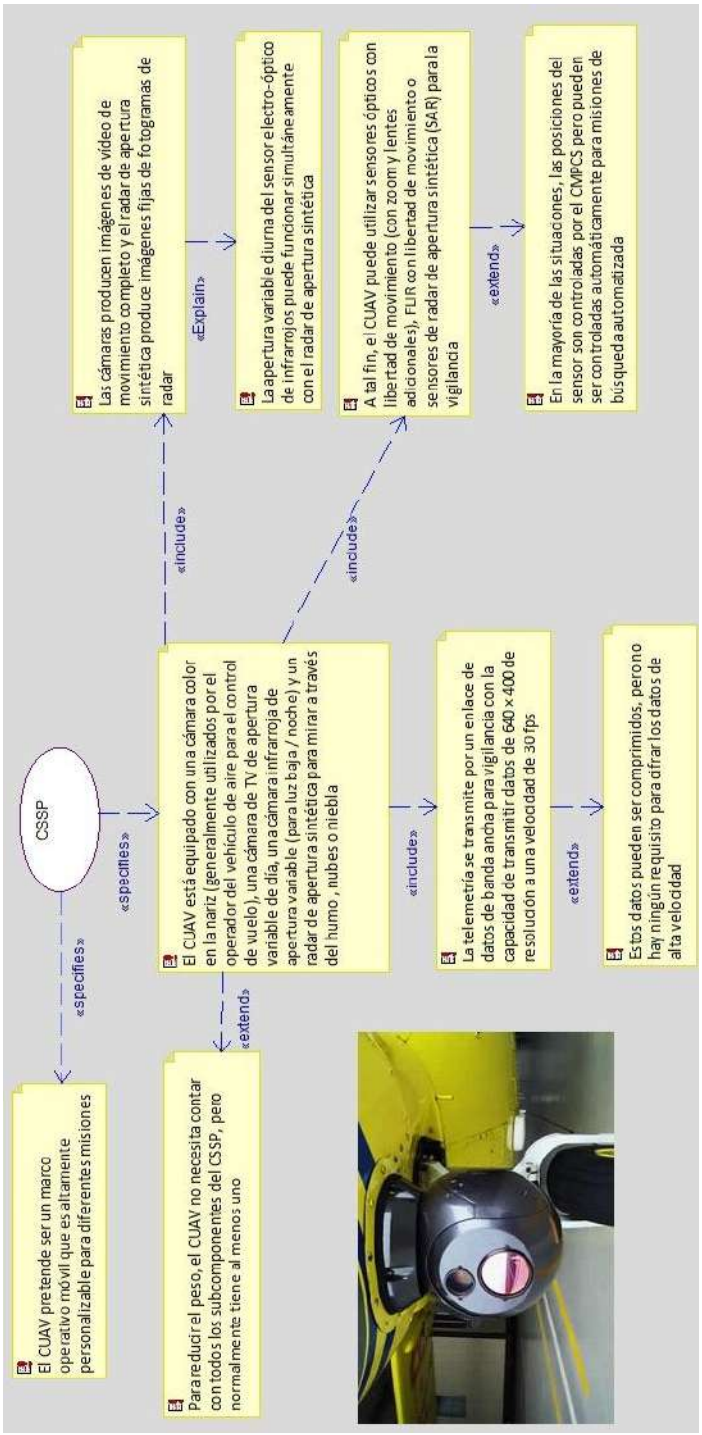
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:_____ Ciclo:_____**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



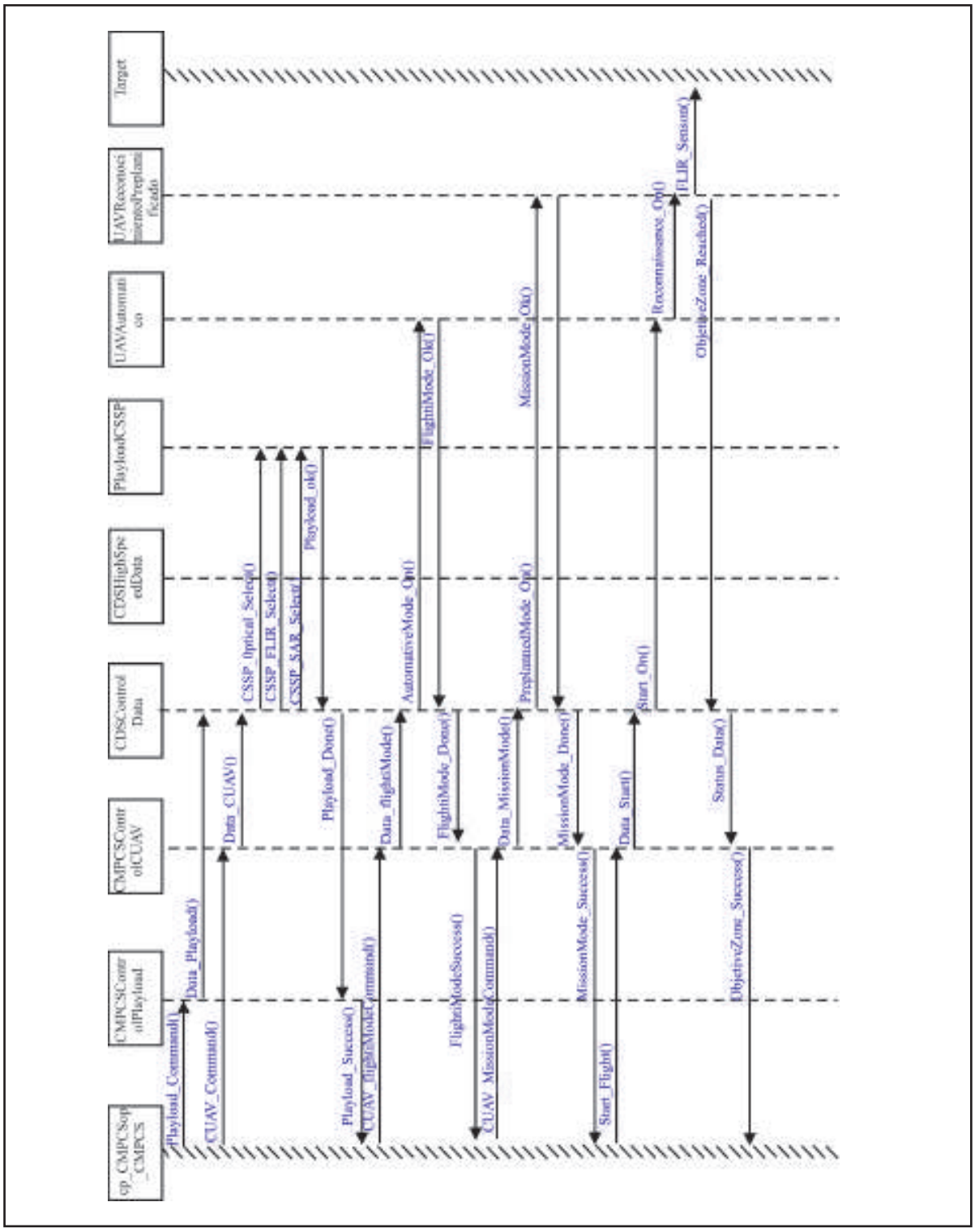
Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:_____ Ciclo:_____**

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



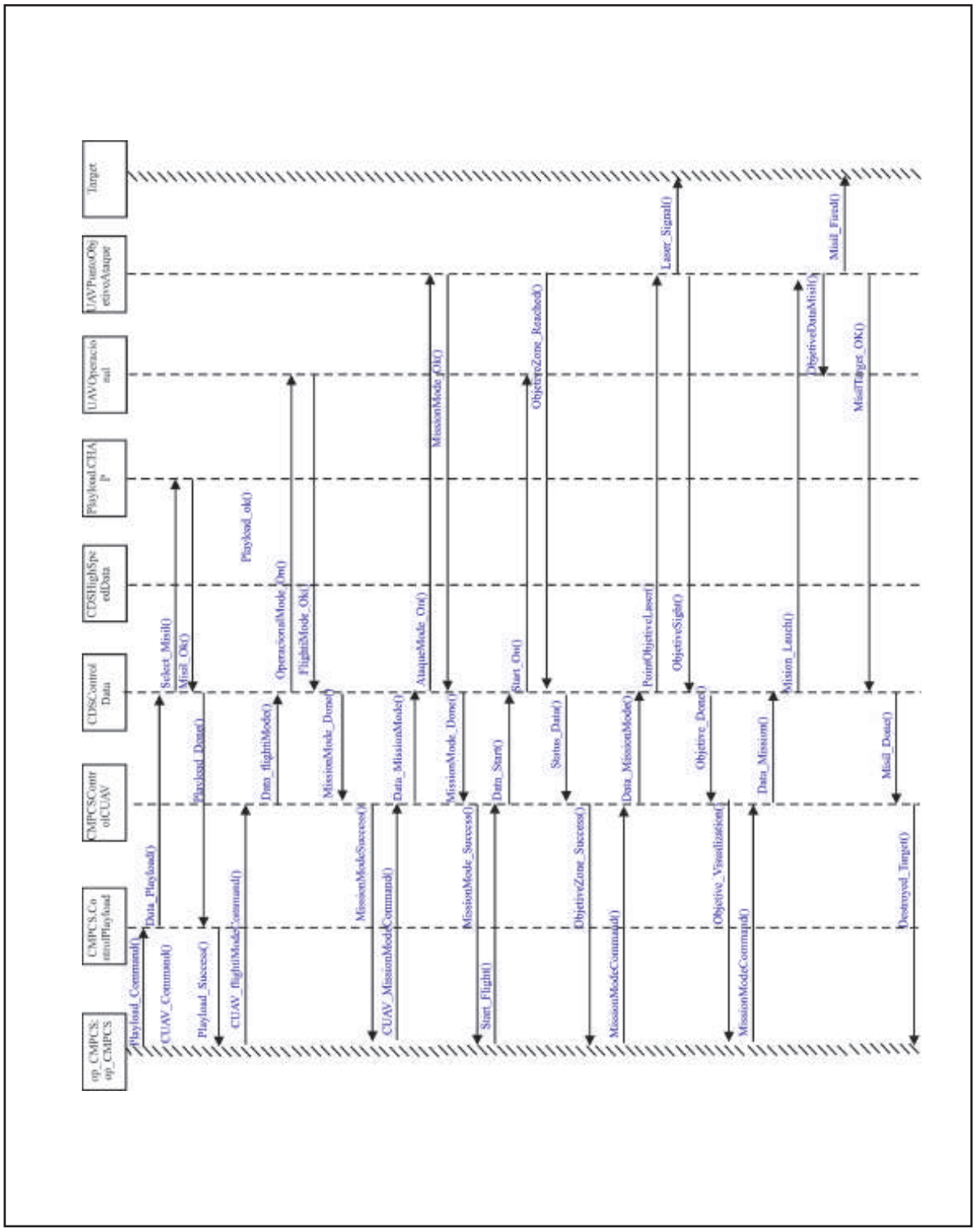
Artefacto – **Diagrama secuencia** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



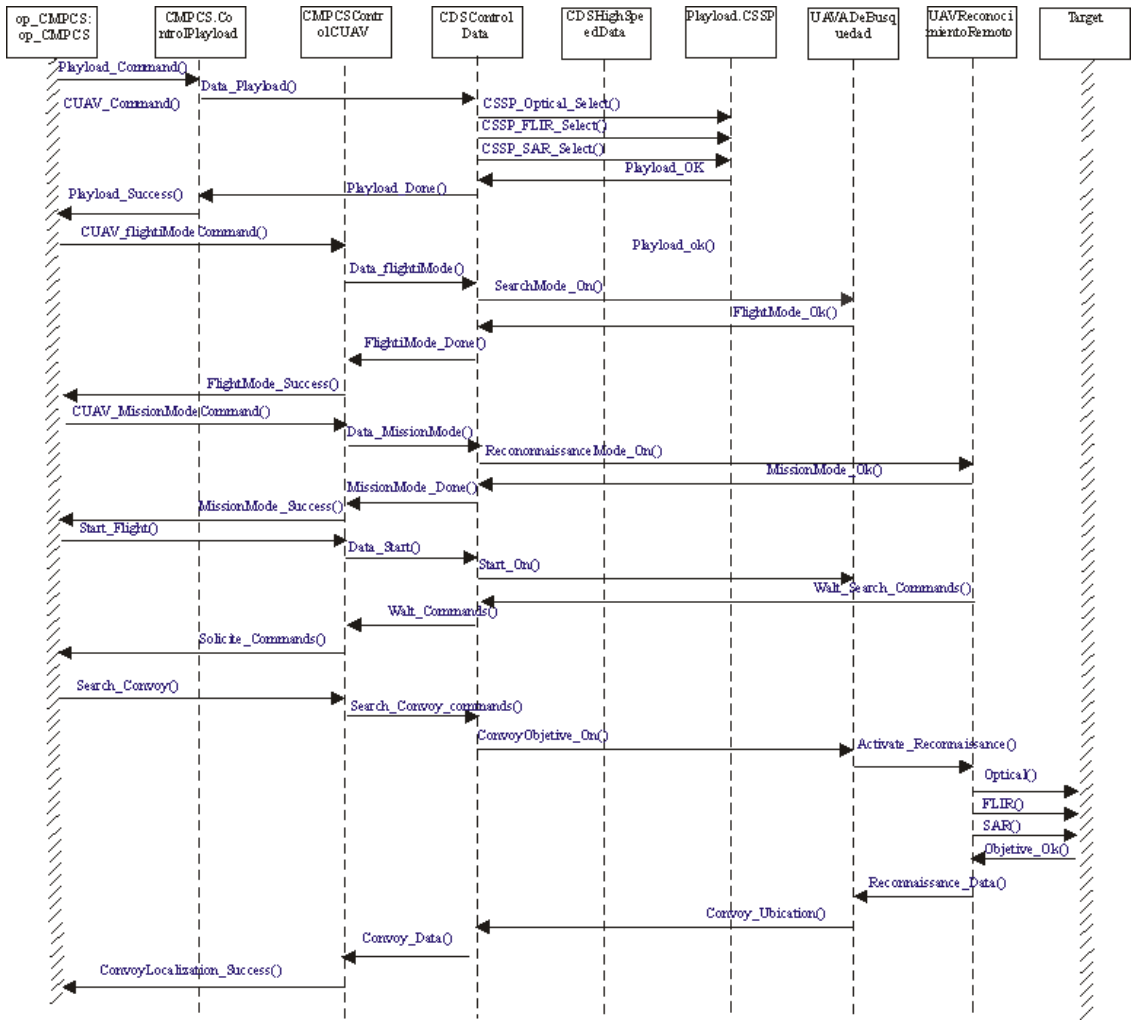
Artefacto – **Diagrama secuencia** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



Artefacto – **Diagrama secuencia** **Fecha:**_____ **Ciclo:**_____

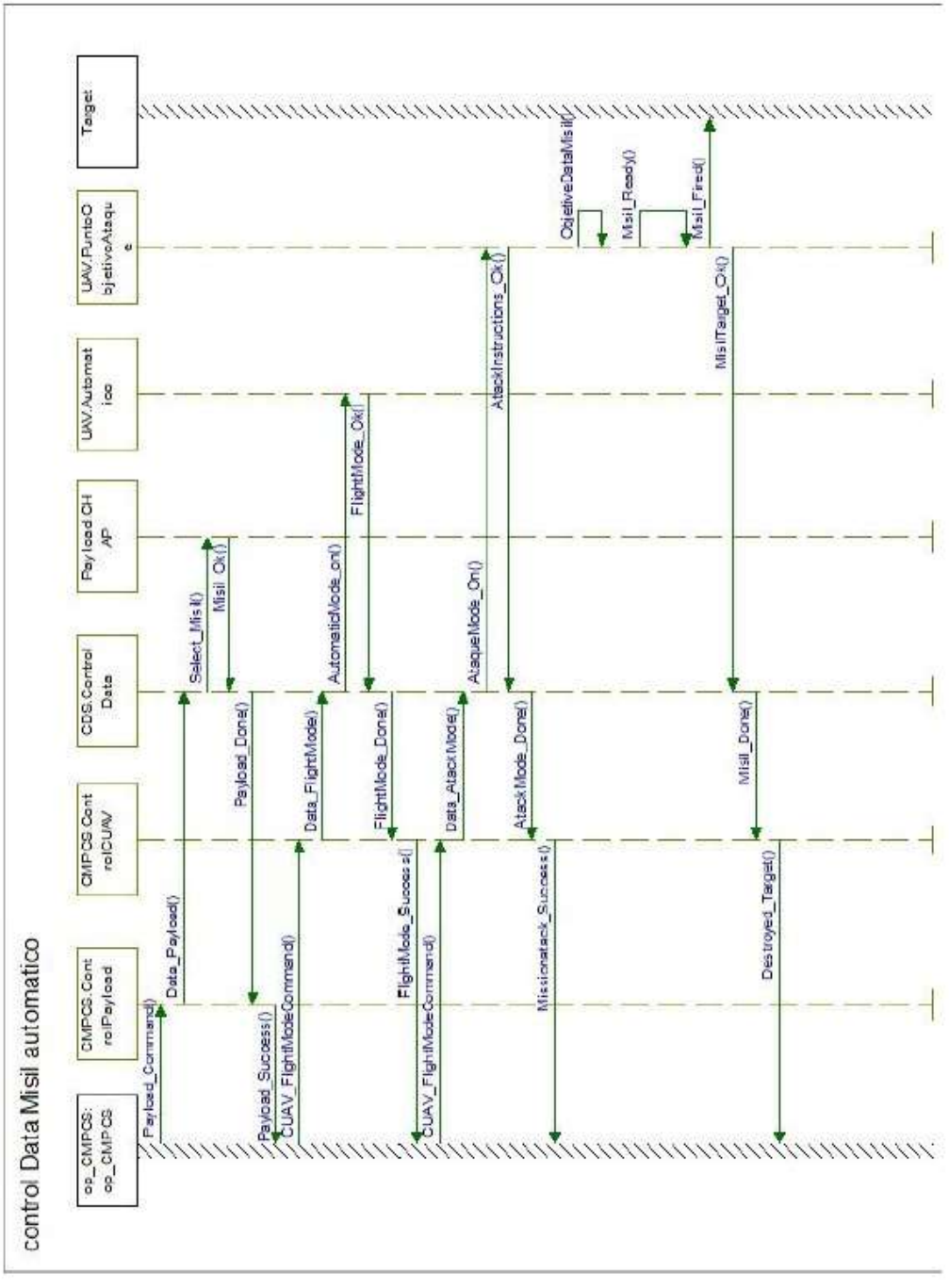
Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



Artefacto – **Diagrama secuencia** **Fecha:_____** **Ciclo:_____**

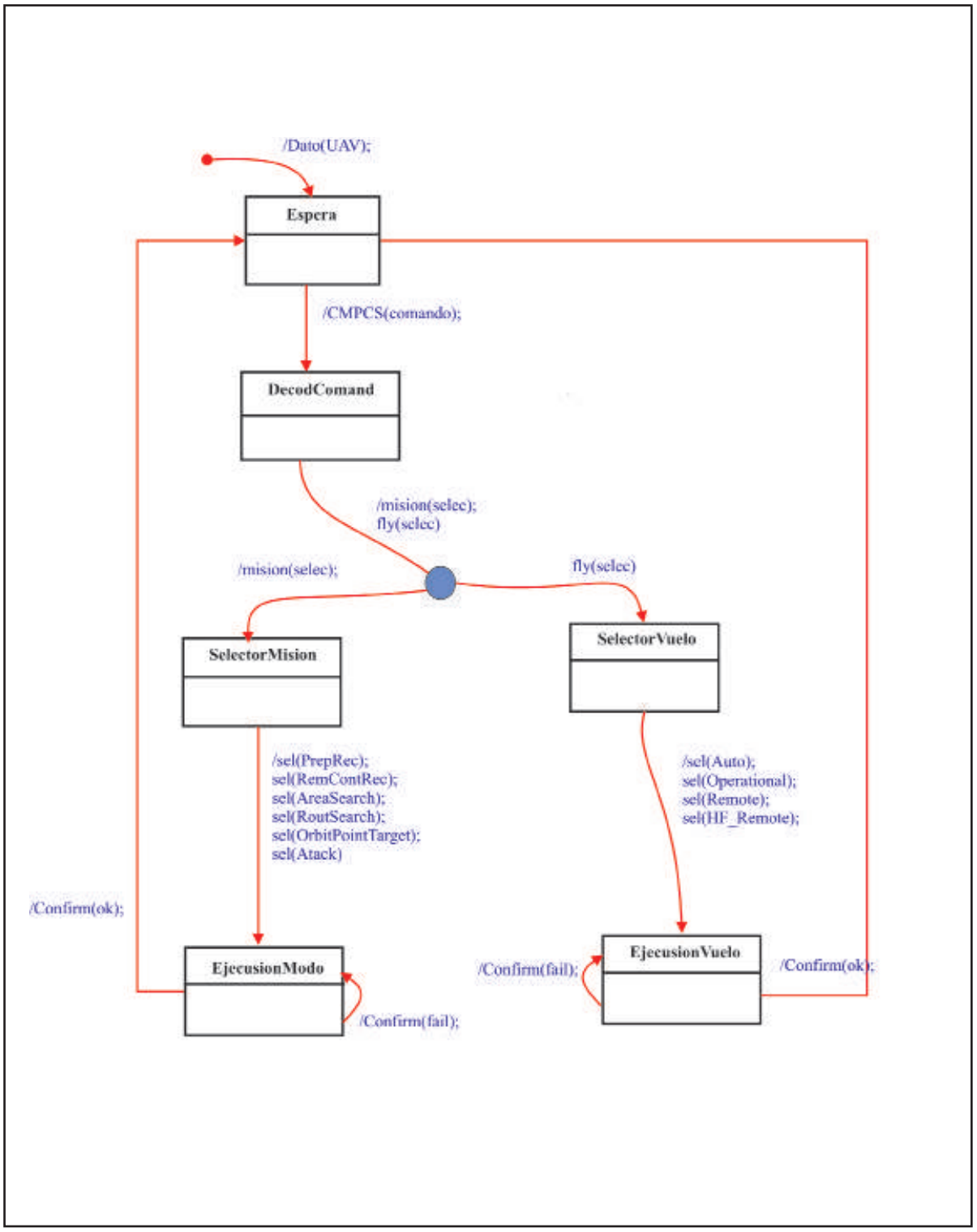
Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López

Diagrama de secuencia del Escenario de Ataque en modo de vuelo Automático



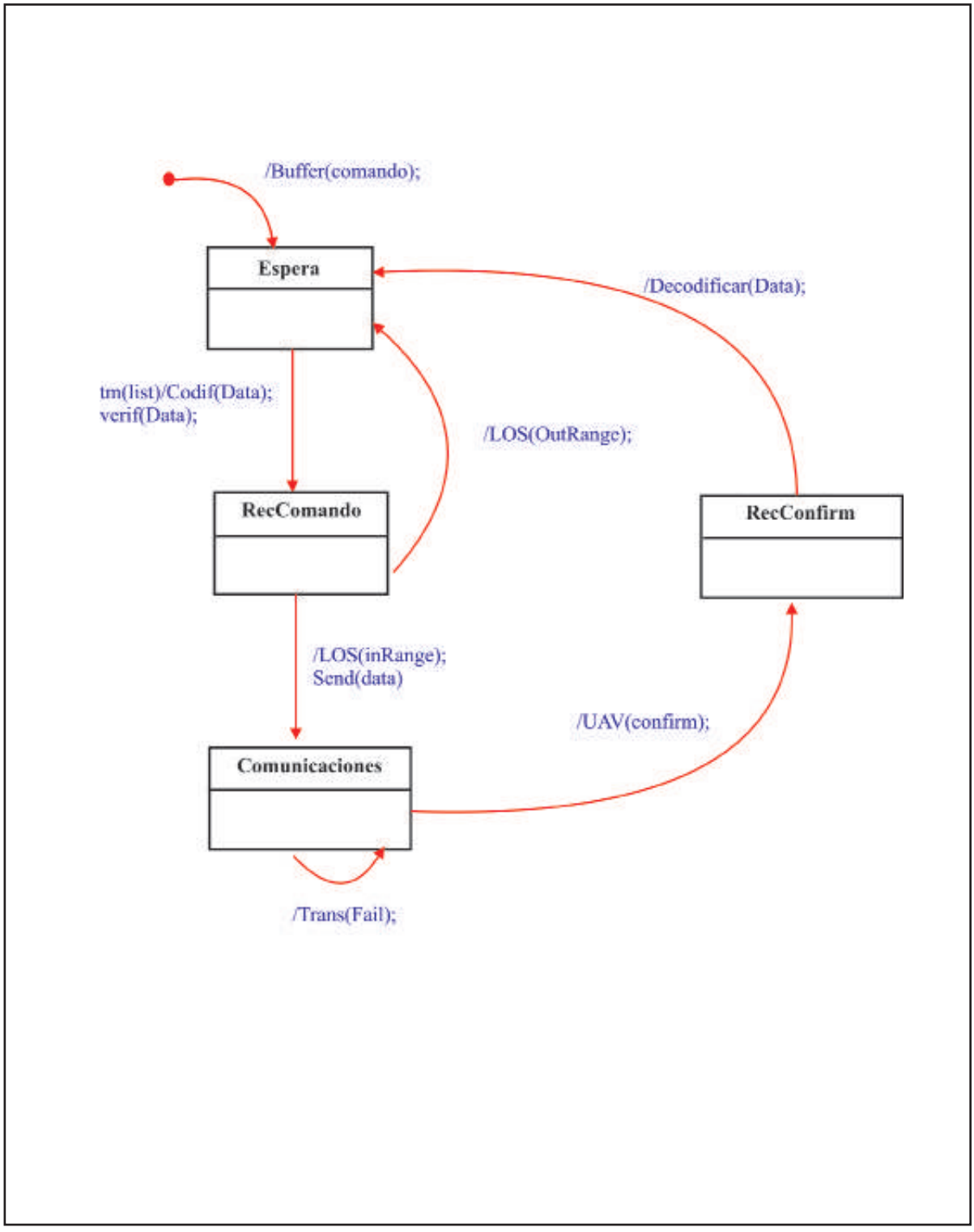
Artefacto – **Máquinas de estado** **Fecha:** _____ **Ciclo:** _____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



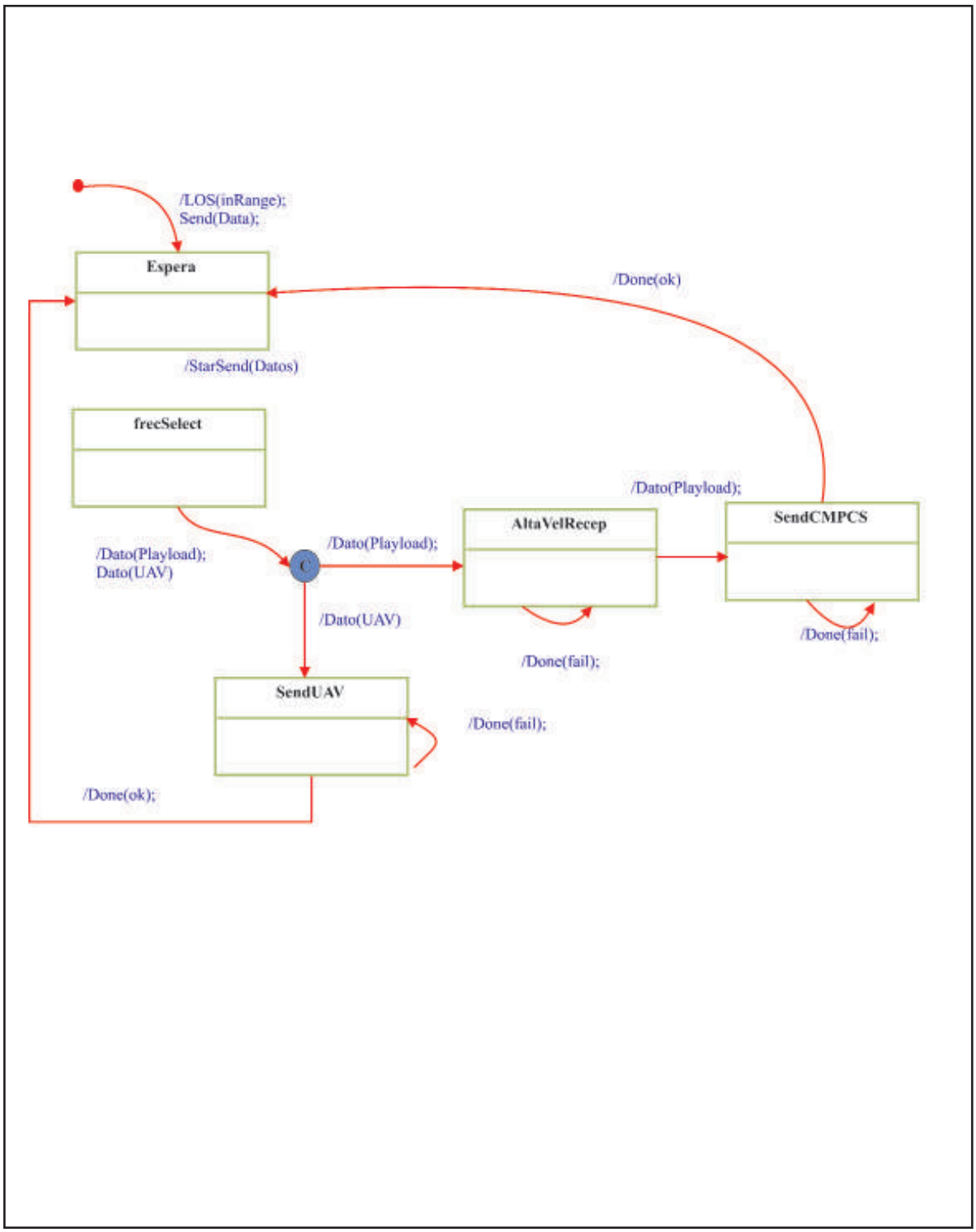
Artefacto – **Máquinas de estado** **Fecha:** _____ **Ciclo:** _____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



Artefacto – **Máquinas de estado** **Fecha:** _____ **Ciclo:** _____

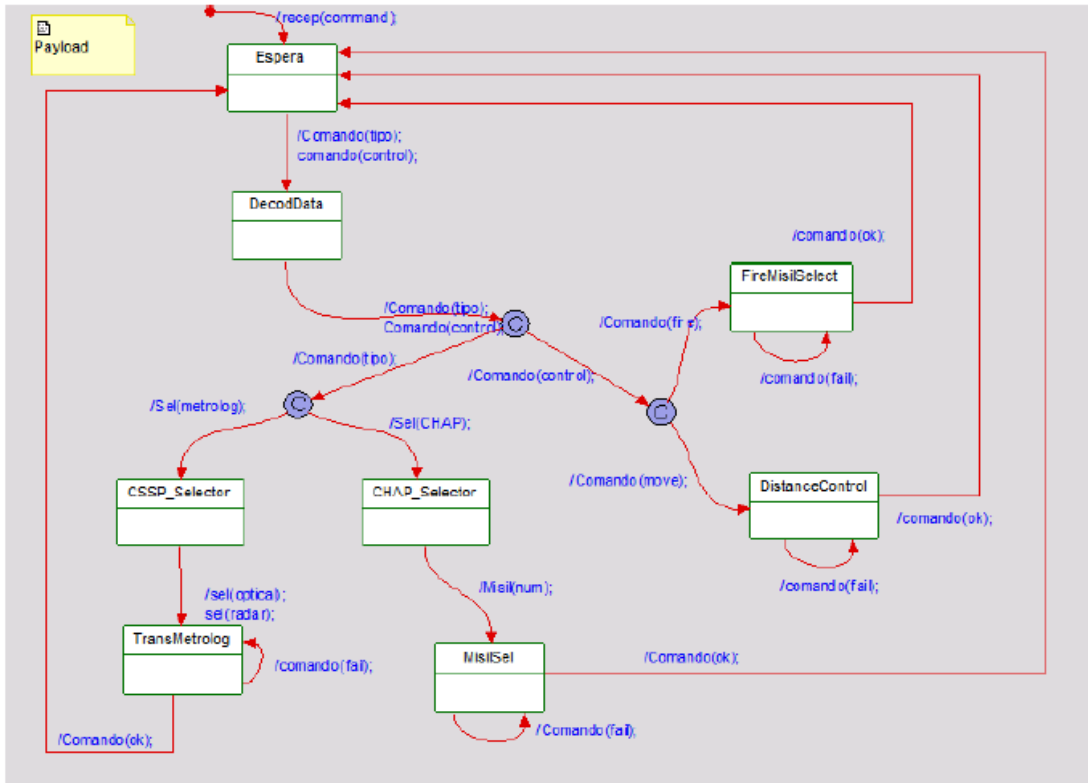
Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



Artefacto – **Máquinas de estado** **Fecha:** _____ **Ciclo:** _____

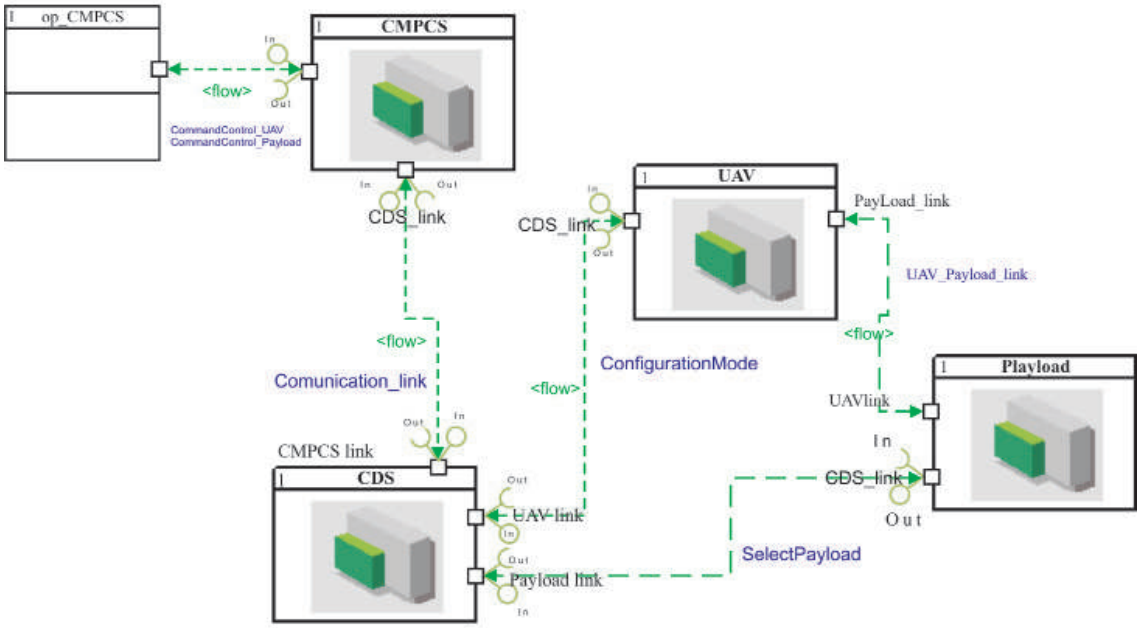
Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López

Coyote Payload



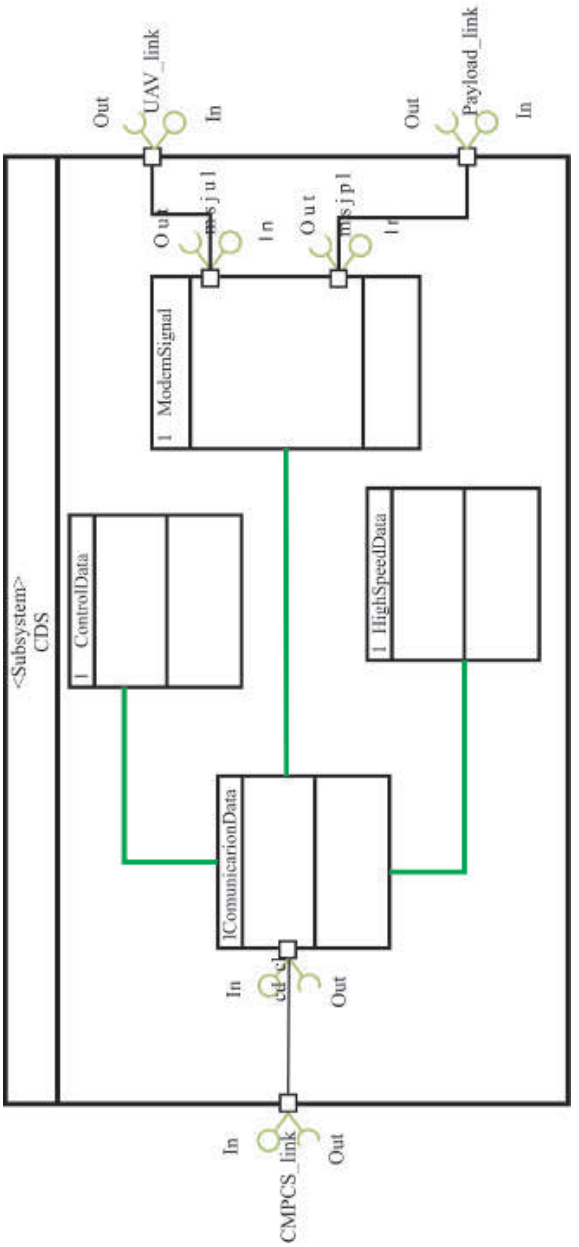
Artefacto – Diagrama de estructura **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



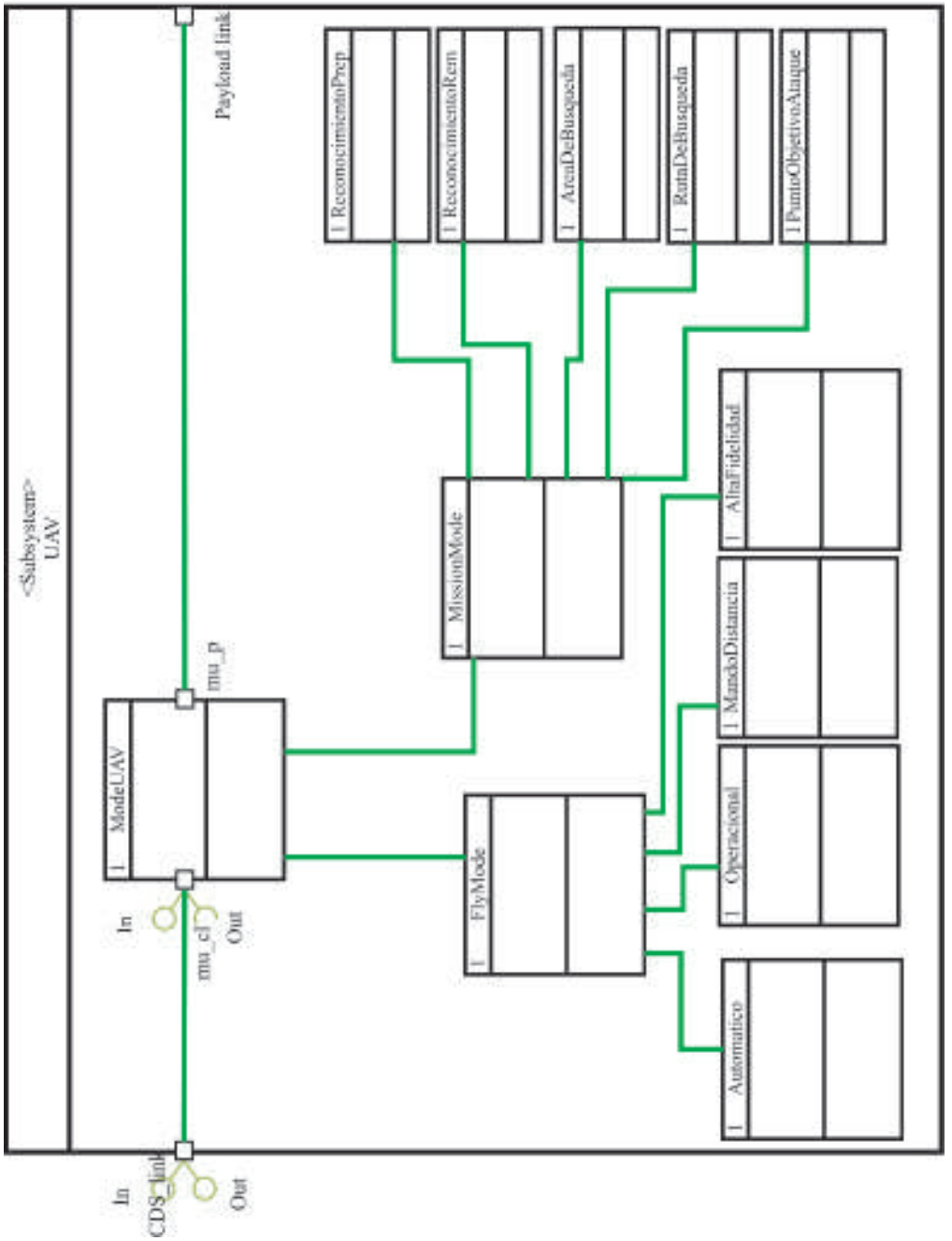
Artefacto – Diagrama de estructura **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



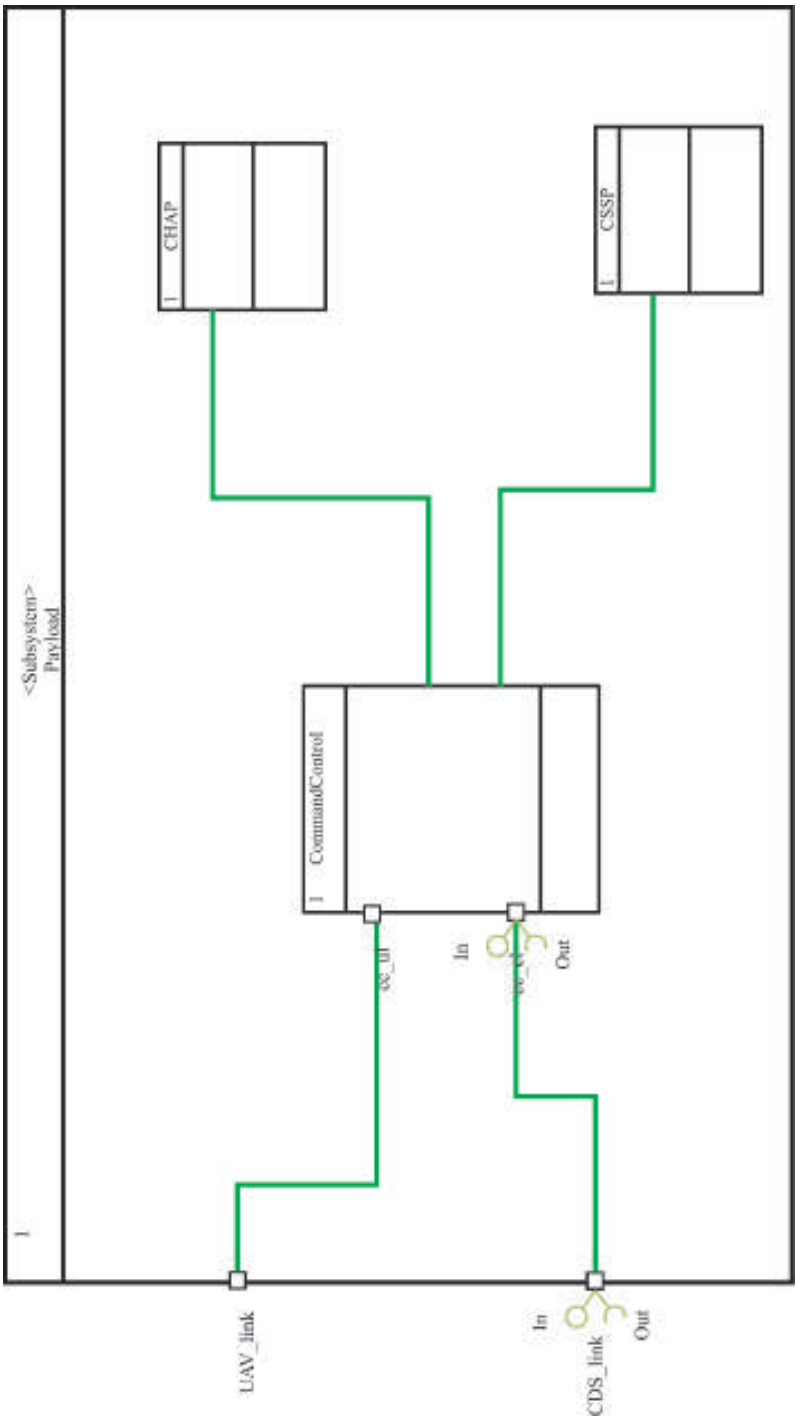
Artefacto – Diagrama de estructura **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



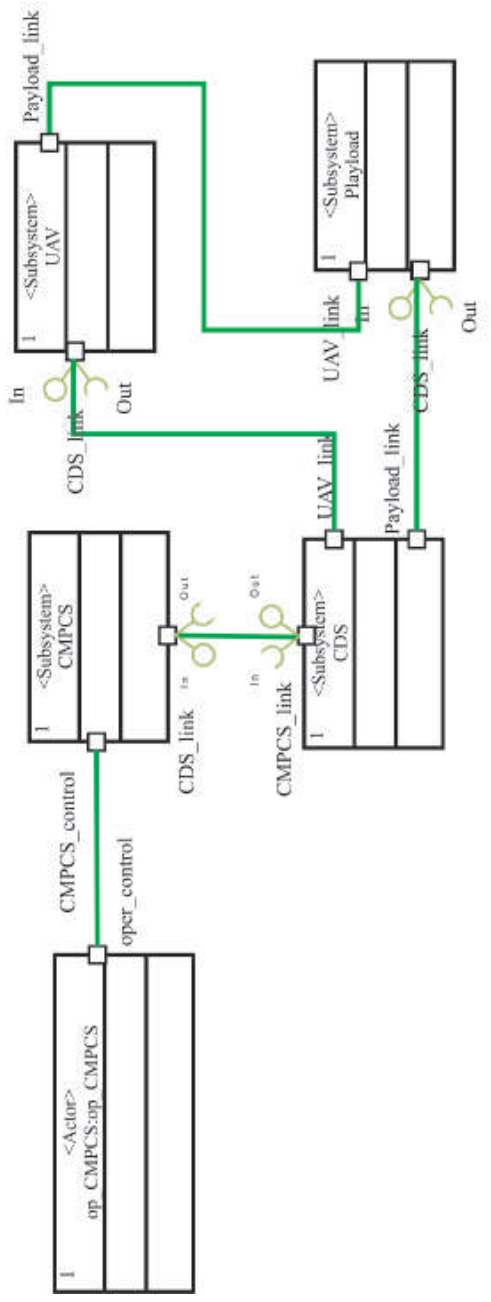
Artefacto – Diagrama de estructura **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



Artefacto – Diagrama de estructura **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	The Coyote Unmanned Air Vehicle System
Nombre del responsable:	Selene Alvarado Legaria, Edel F. Cuevas López



En este proyecto se llevaron a cabo las etapas señaladas en la Figura 6.25. La fase de inicio fue breve y comenzó con una visión común inicial de los objetivos del proyecto. A partir de las especificaciones del sistema se obtuvieron más de 90 requisitos, así, el crecimiento de la complejidad en los requisitos funcionales y extra funcionales del sistema muestra la importancia de utilizar SPIES. Para el diseño del sistema, los requisitos se clasificaron de acuerdo a las siguientes definiciones:

- **Requisito operacional:** es aquel que especifica cómo colabora el sistema con otros elementos (actores) en su entorno.
- **Requisito funcional:** son las condiciones que tienen que ver con el comportamiento del sistema: características, capacidades y seguridad.
- **Requisito de diseño:** son las características que el diseño del sistema debe considerar, pero no son características propias del sistema.
- **Restricción:** se denominan comúnmente "restricciones" para remarcar su influencia restrictiva (por lo general en su enunciado se incluye la palaba "Debe").

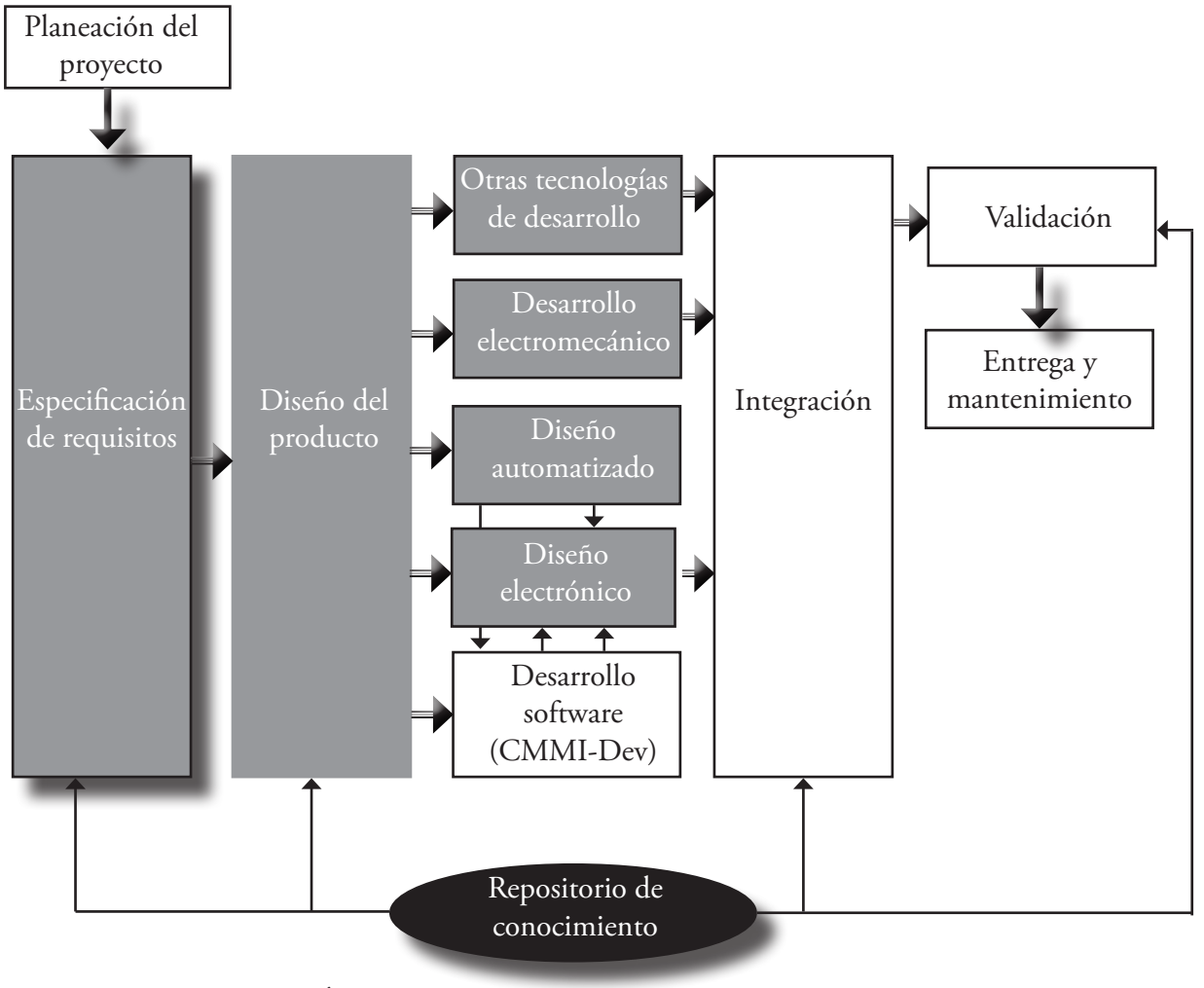


Figura 6.25. Áreas de proceso de SPIES abarcadas por el equipo de diseño del SISTEMA THE COYOTE UNMANNED AIR VEHICLE SYSTEM

Por otra parte para los casos de uso el equipo consideró los siguientes puntos:

- Los Casos de Uso deben regresar al menos un valor para un actor.
- Un Caso de Uso debe contener al menos tres escenarios, cada escenario debe consistir de múltiples mensajes actores-sistemas.

Ahora bien, para llevar a cabo el diseño del sistema siguiendo SPIES, el equipo realizó los diagramas de secuencia, los diagramas de máquinas de estado y de estructura. Los diagramas de secuencia fueron útiles para presentar los eventos que se generan en un caso de uso para un escenario específico. Mientras que los diagramas de estructura definieron la estructura del sistema y los diagramas máquinas de estado indican los estados a los que puede entrar (sobre una línea de tiempo) un actor, un caso de uso o una clase y los mensajes, eventos u operaciones que causan la transición de un estado a otro.

6.3. SISTEMA DE ENTRENAMIENTO DREAMBLUE

El siguiente SE no partió de un conjunto de especificaciones como los anteriores, sino de un enunciado, que expresa el objetivo a alcanzar por un trabajo tesis. El objetivo general se enunció como sigue: "Diseñar y desarrollar un sistema educativo como herramienta de entrenamiento del protocolo de comunicaciones Bluetooth utilizando la metodología SPIES". Ahora bien, a partir del objetivo, se identificó el siguiente conjunto inicial de requisitos del sistema:

- El sistema DreamBlue debe constar de dos nodos comunicados mediante el protocolo de comunicaciones Bluethooth.
- Los nodos de comunicación Bluetooth debe ser configurables.
- Cada nodo debe constar de un Equipo Terminal de Datos, como interfaz con el usuario, y un módulo Bluetooth que realiza la conexión para formar una picored[8].
- En base en los módulos desarrollados, se debe configurar una picored.

En la Figura 6.26, se presenta el diagrama de requisitos general del sistema diseñado por el Tesista para capturar los requisitos del SE de forma efectiva y práctica. Por lo que, de forma gráfica presenta la taxonomía de los requisitos de la tesis a un nivel de abstracción alto. A partir de este diagrama se puede diseñar el Diagrama de Casos de Uso.

El diagrama de caso de uso general del sistema (véase Figura 6.27) se forma por los actores: Usuario y Picored. En el DCU general se observa que el sistema cumple con uno de los requisitos principales del sistema (el sistema DreamBlue debe constar de dos nodos comunicados mediante el protocolo de comunicaciones Bluethooth) y la relación que debe existir entre los actores. Ahora bien, DCU general fue descompuesto en casos de uso más pequeños para comprender mejor el sistema y saber cómo hacer las cosas. En la Figura 6.28, se presenta el diagrama de casos de uso para el nodo del sistema (componente principal del SE), así como el mapeo de requisitos.

Cada nodo consta de un Equipo Terminal de Datos (DTE) como interfaz con el usuario (UI) y un módulo Bluetooth, o Equipo de Terminación del Circuito de Datos (DCE), que realiza la conexión Bluetooth para formar una picored. La comunicación entre el módulo DTE y el módulo DCE se lleva a cabo por medio del protocolo RS232. Todo lo anterior esta señalado en el DCU de la Figura 6.28.

8 La formación de una red con dos o más dispositivos Bluetooth en un rango de cobertura se conoce como picored

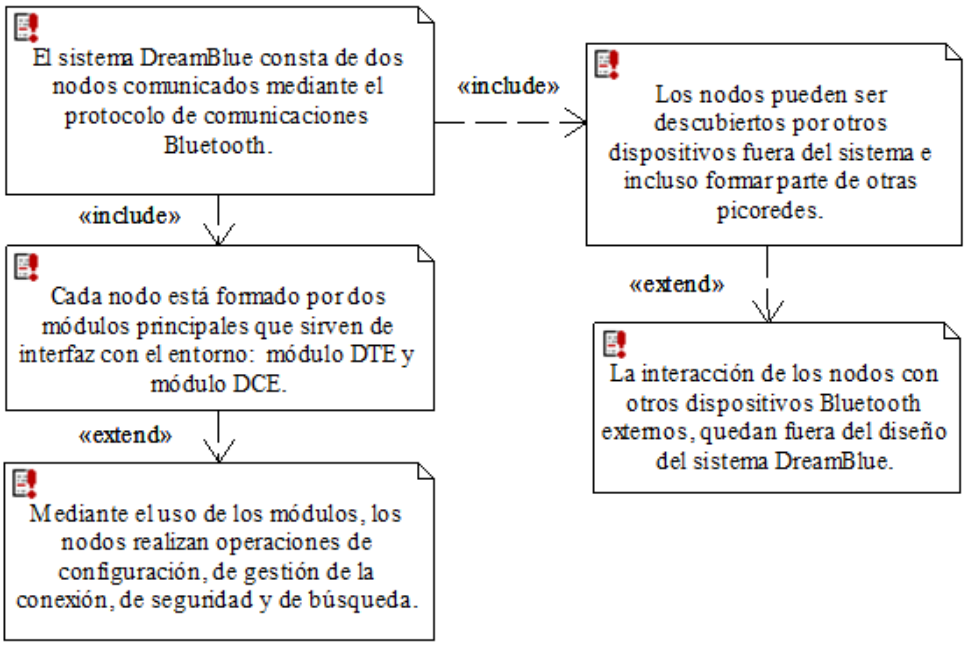


Figura 6.26. Diagrama de requerimientos generales del sistema DreamBlue

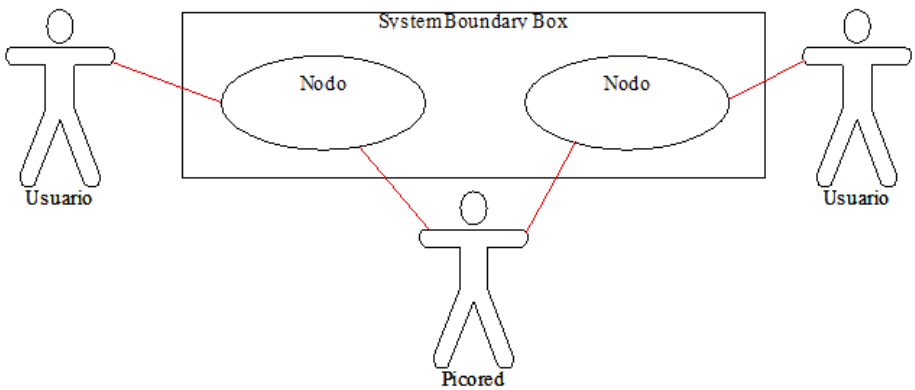


Figura 6.27. Diagrama de caso de uso general para el sistema DreamBlue

Este tipo de diagramas es de gran ayuda para los desarrolladores, pues les ayuda a considerar los elementos que integrarán al sistema, tanto en hardware como en software. Bajando un nivel de abstracción se presenta la Figura 6.29, que describe el módulo DTE. El módulo DTE se integra de dos interfaces para comunicarse con su entorno:

- Interfaz de usuario del módulo DTE (UI_DTE): Establece la comunicación entre el nodo y el usuario.
- Interfaz RS232 del módulo DTE (COM_DTE): Comunica y acopla al módulo DTE con el módulo DCE.

Por otra parte, en la Figura 6.29 se presenta el DCU para la Interfaz de usuario del módulo DTE (UI_DTE).

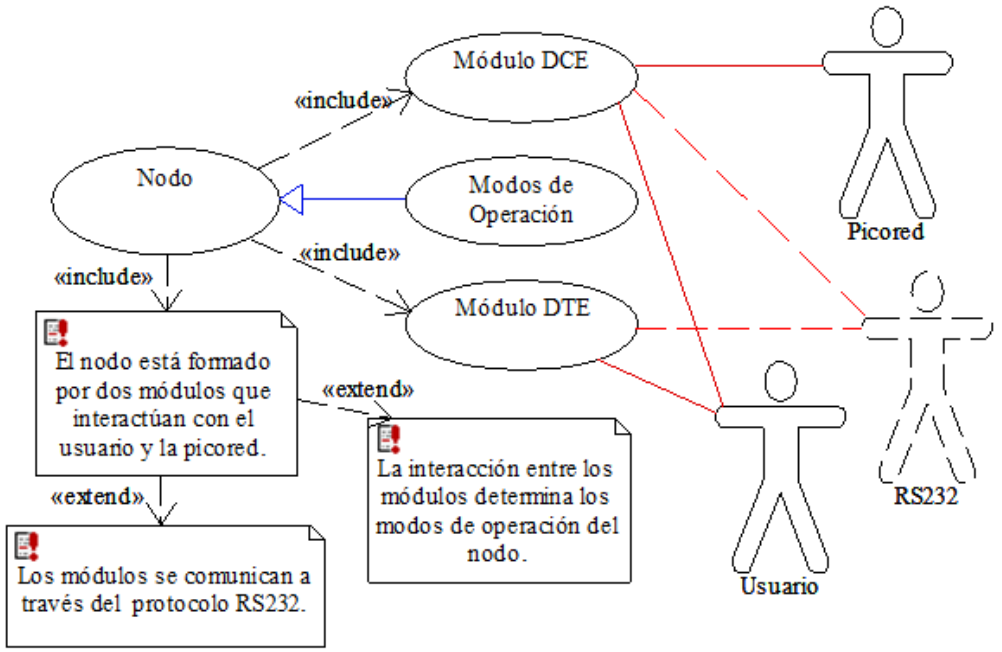


Figura 6.28. Diagrama de caso de uso para nodo del sistema DreamBlue

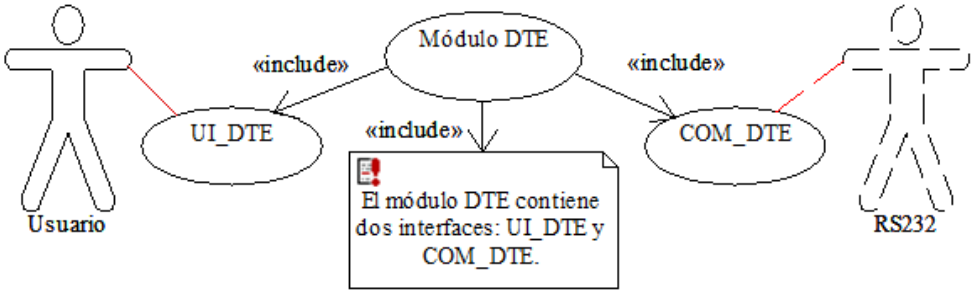


Figura 6.29. Diagrama de caso de uso para el módulo DTE

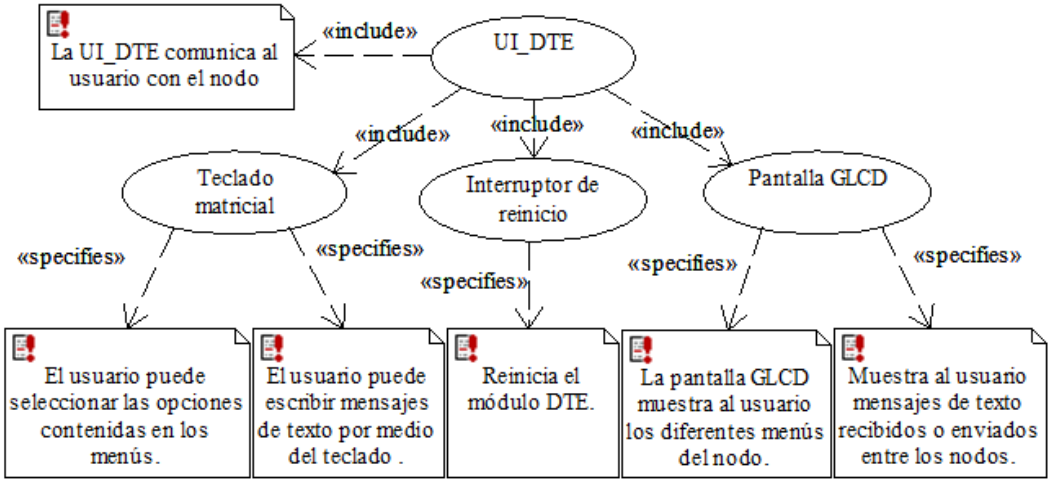


Figura 6.30. Diagrama de caso de uso para la UI_DTE

En las siguientes Figuras se presentan los diagramas de casos de uso para COM_DTE y para el módulo DTE, de igual forman se desarrollaron los diagramas para los componentes y modos de operación: UI_DCE, LED RGB, COM_DCE, CINP, modo de órdenes AT, Configuración del Nodo, Configuración RS232, Modo Conectable, Modo de Recepción, Modo de Transmisión, Modo Sondeo, Modo Detectable, Modo Seguridad, Modo de Datos, Modo Servidor, Modo Cliente. También se diseño el Diagrama de caso de uso para la generalización Nombre y Clase, para Restaurar Nodo, Modo Ahorro, Gestión de la Conexión, Gestión de Mensajes, Servicios Suplementarios. En total, se realizaron 30 diagramas de casos de uso para este sistema, el cual es el número de DCU ideal para que el desarrollador muestre el nivel de abstracción y entendimiento necesario para desarrollar y documentar su SE.

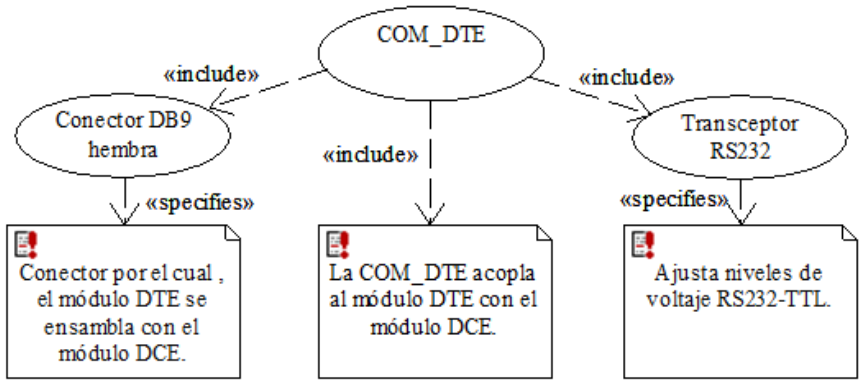


Figura 6.31. Diagrama de caso de uso para COM_DTE

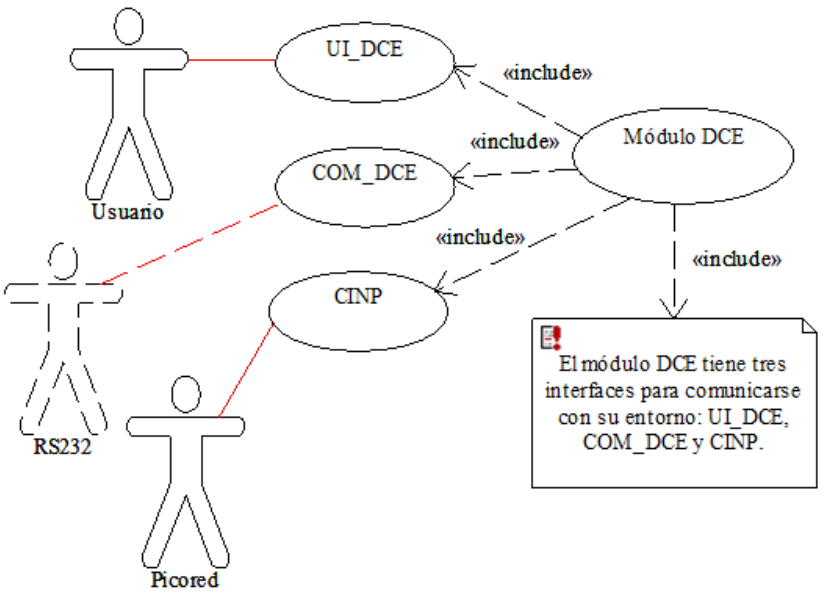


Figura 6.32. Diagrama de caso de uso para el módulo DTE

Como parte del diseño se realizaron los diagramas de estructura. Con este tipo de diagramas se capturó la estructura interna del sistema e identificaron las piezas que lo componen. En la Figura 6.33 se muestra los subsistemas o elementos que integran un nodo, estos son: Módulo DTE, Módulo DCE, Modos de Operación, Modo de Datos y Modo de Órdenes AT. Los módulos DTE y DCE controlan las operaciones que realizan los actores, mientras que, los módulos de Modo de Órdenes AT y Modo de Datos controlan el acceso a los subsistemas respectivos a cada modo de operación.

En la Figura 6.35 se presenta el diagrama generado para la estructura general de cada nodo DreamBlue, en el se muestra: los objetos (las entidades que encapsulan estado y comportamiento), los puertos (para especificar las interfaces entre los componentes del sistema), y el flujo de información entre los elementos. Este diagrama sirve de guía ya que en él se describe el sistema al mostrar los elementos, la interacción que existe entre el nodo, los actores, entre otros cosas.

Ahora bien, el diagrama general de la Figura 6.36 se puede describir en mayor detalle al diseñar el diagrama de estructura de cada componente, por ejemplo, en la Figura 6.34 se presenta el diagrama de estructura del subsistema Configuración del Nodo. De igual forma se realizaron los diagramas de estructura del subsistema Gestión de la Conexión, Servicios Suplementarios y Gestión de Mensajes.

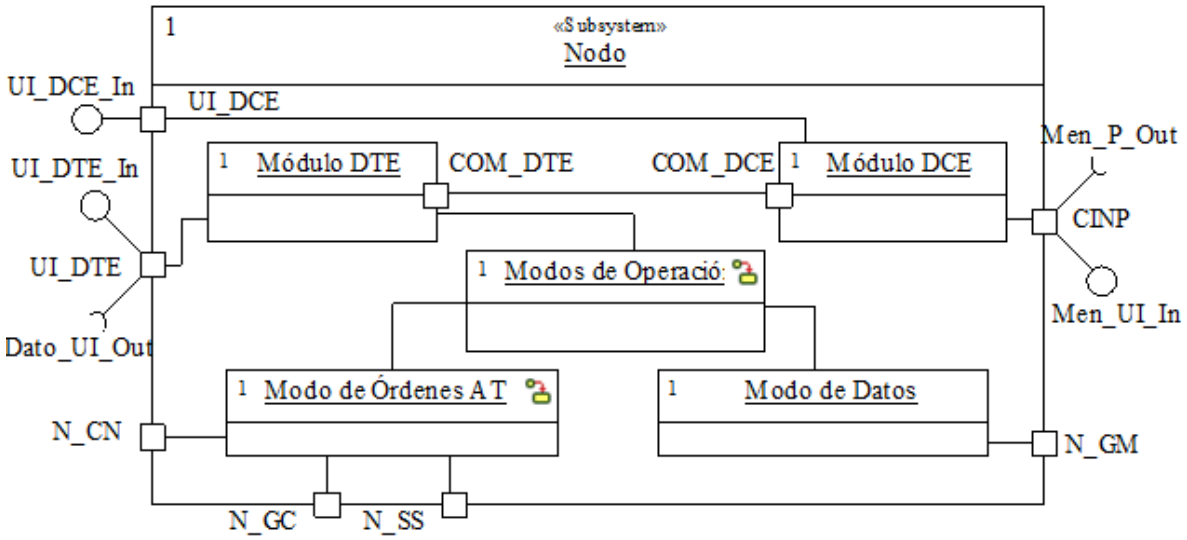


Figura 6.33. Diagrama de estructura del nodo

Con los diagramas de secuencia se comprende mejor lo que se está construyendo; al resaltar la ordenación temporal de los mensajes que se intercambian durante un escenario concreto. En la Figura 6.37, se presenta el diagrama de secuencia diseñado para el escenario acceso al modo de órdenes AT; a partir del diagrama es fácil identificar los objetos que intervienen (nodo, picored), la dimensión temporal y el paso de mensajes. Las notas en el diagrama de secuencia ayudan a que sea más fácil de entender la secuencia. En la Figura 6.38 y 6.39 los diagramas de secuencia para los escenarios: acceso al modo de operación de datos y configuración del nodo para establecer los parámetros de configuración RS232 sobre el nodo.

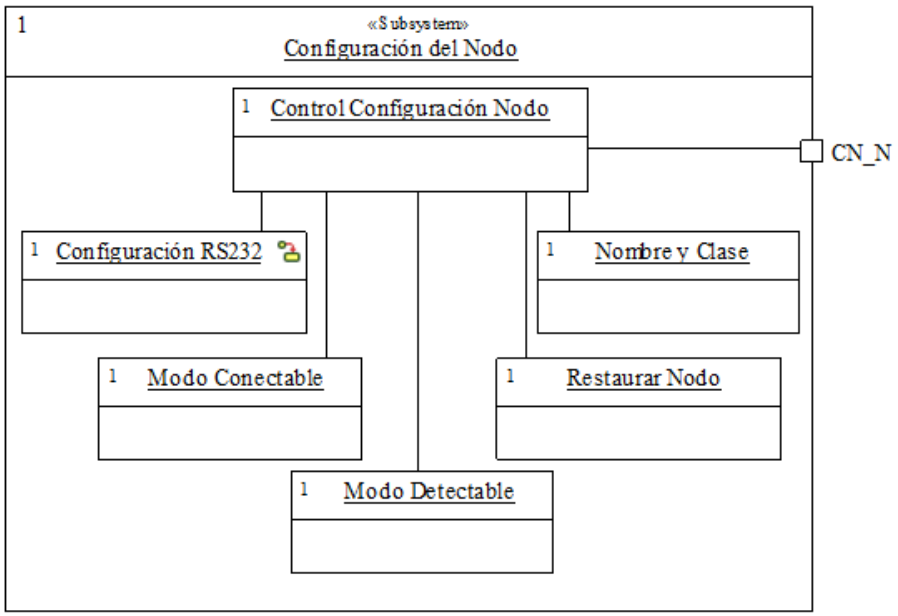


Figura 6.34. Diagrama de estructura del subsistema Configuración del Nodo

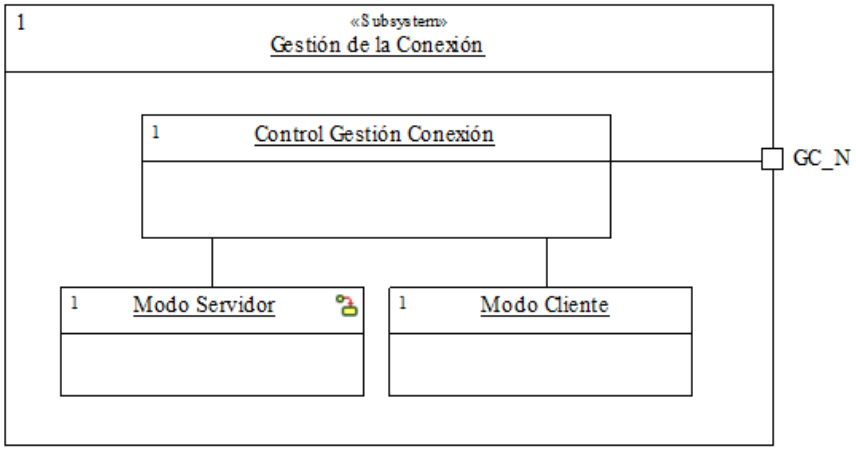


Figura 6.35. Diagrama de estructura del subsistema Gestión de la Conexión

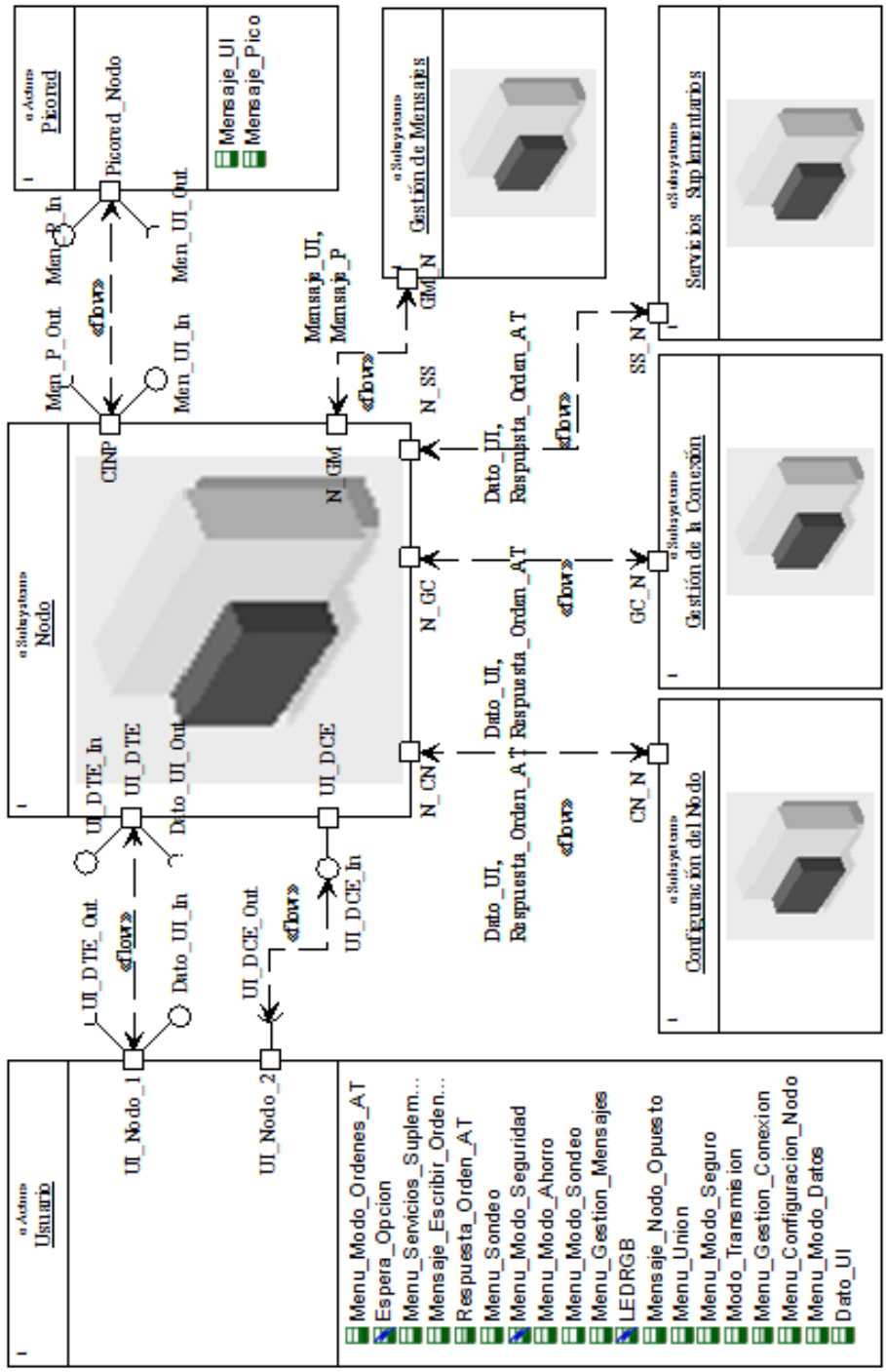


Figura 6.36. Diagrama de estructura general de cada nodo DreamBlue

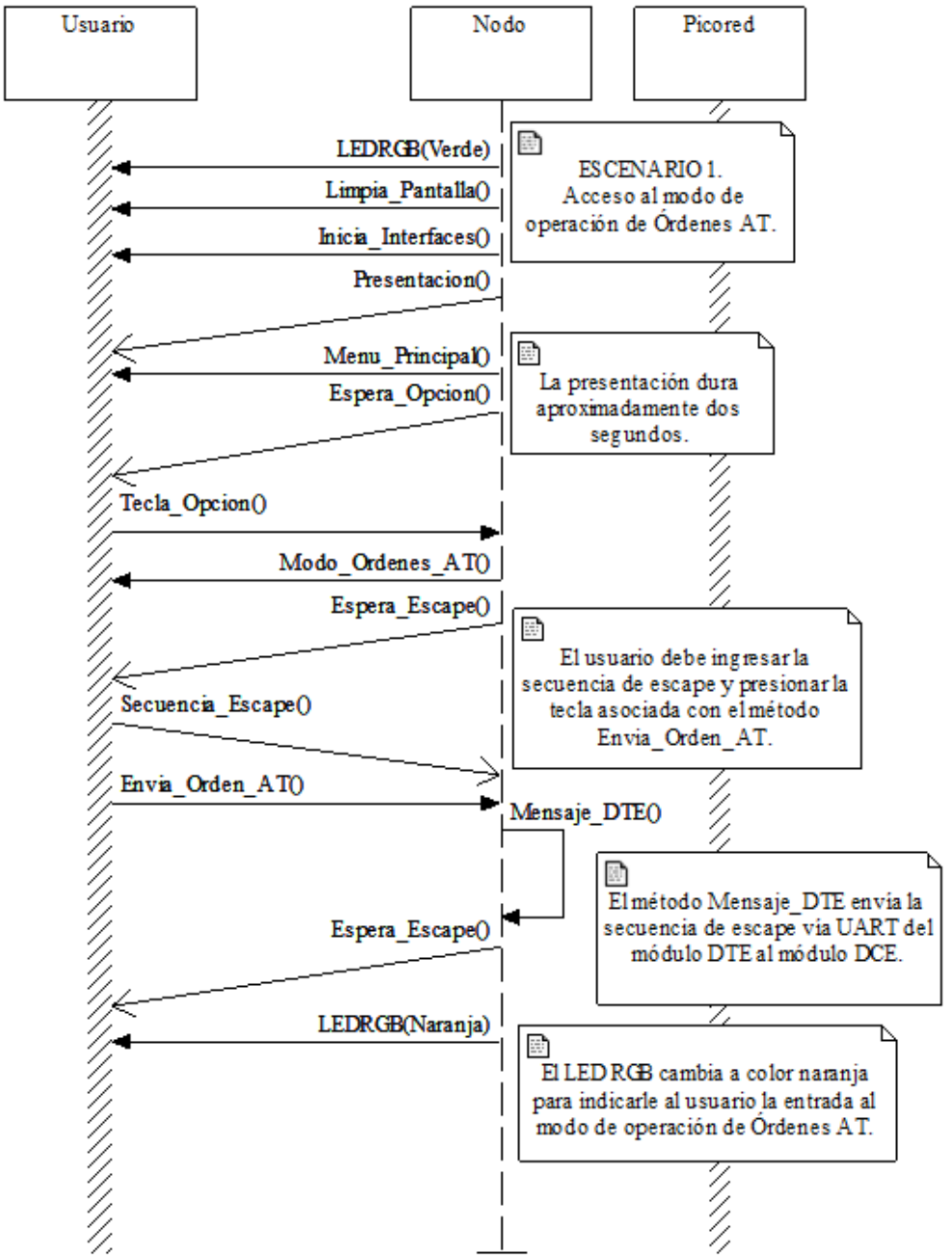


Figura 6.37. Diagrama de secuencia para el Escenario 1: acceso al modo de órdenes AT

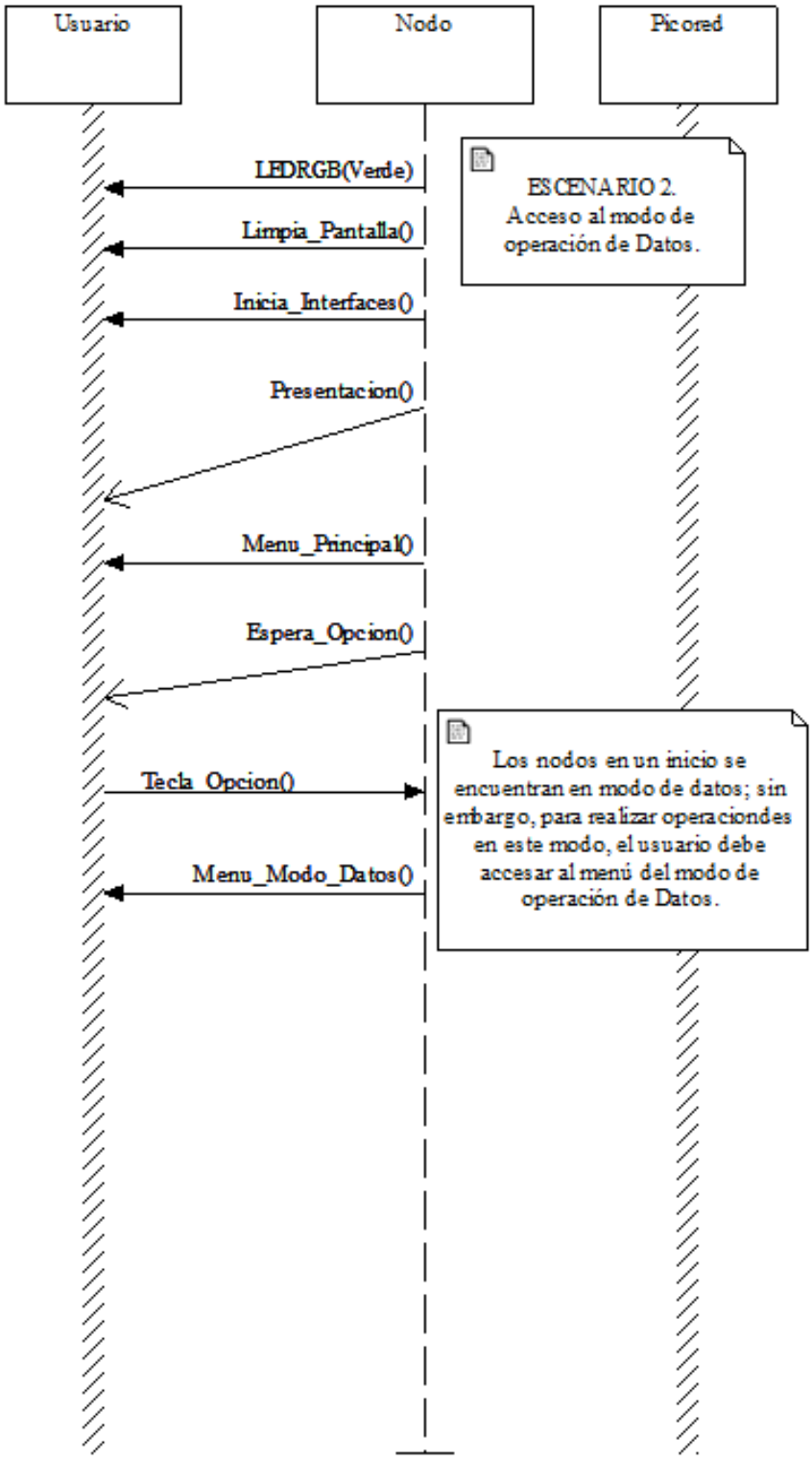


Figura 6.38. Diagrama de secuencia para el Escenario 2: acceso al modo de operación de daros

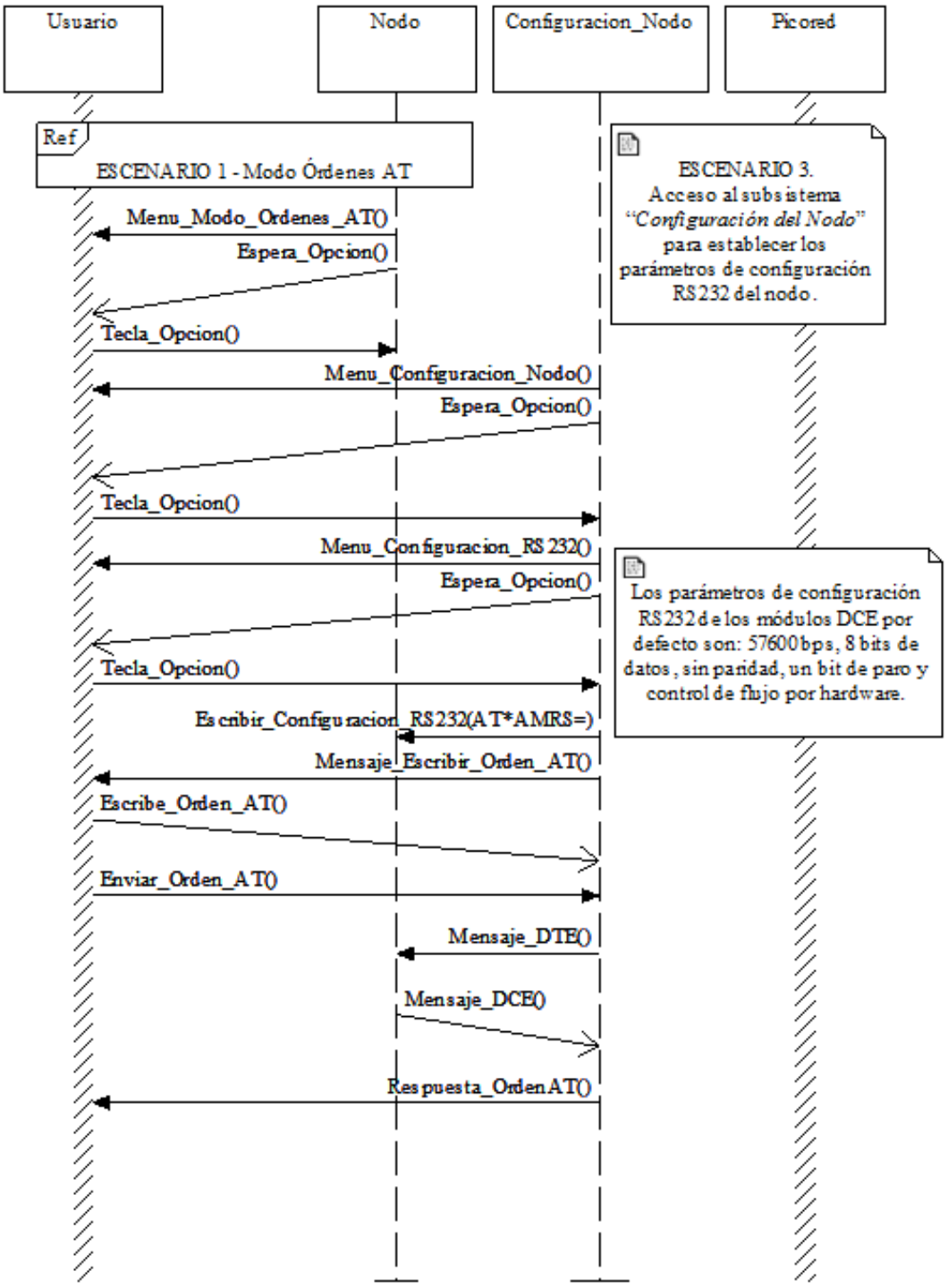


Figura 6.39. Diagrama de secuencia para el Escenario 3: acceso al subsistema "Configuración del nodo" para establecer los parámetros de configuración RS232 del nodo

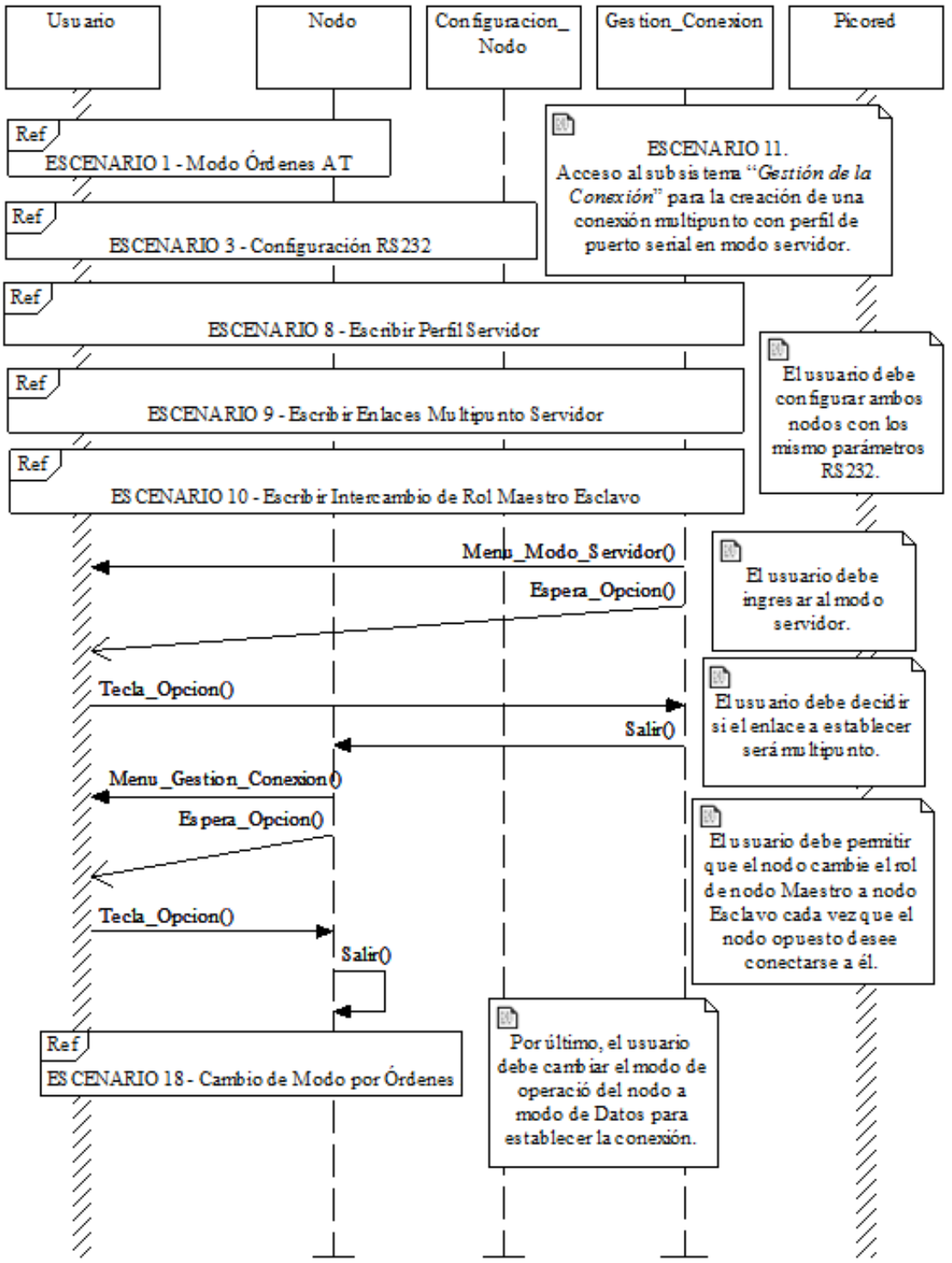


Figura 6.40. Diagrama de secuencia para el Escenario 11: Gestión de la Conexión para la creación de una conexión serial multipunto con perfil de puerto serial en modo servidor

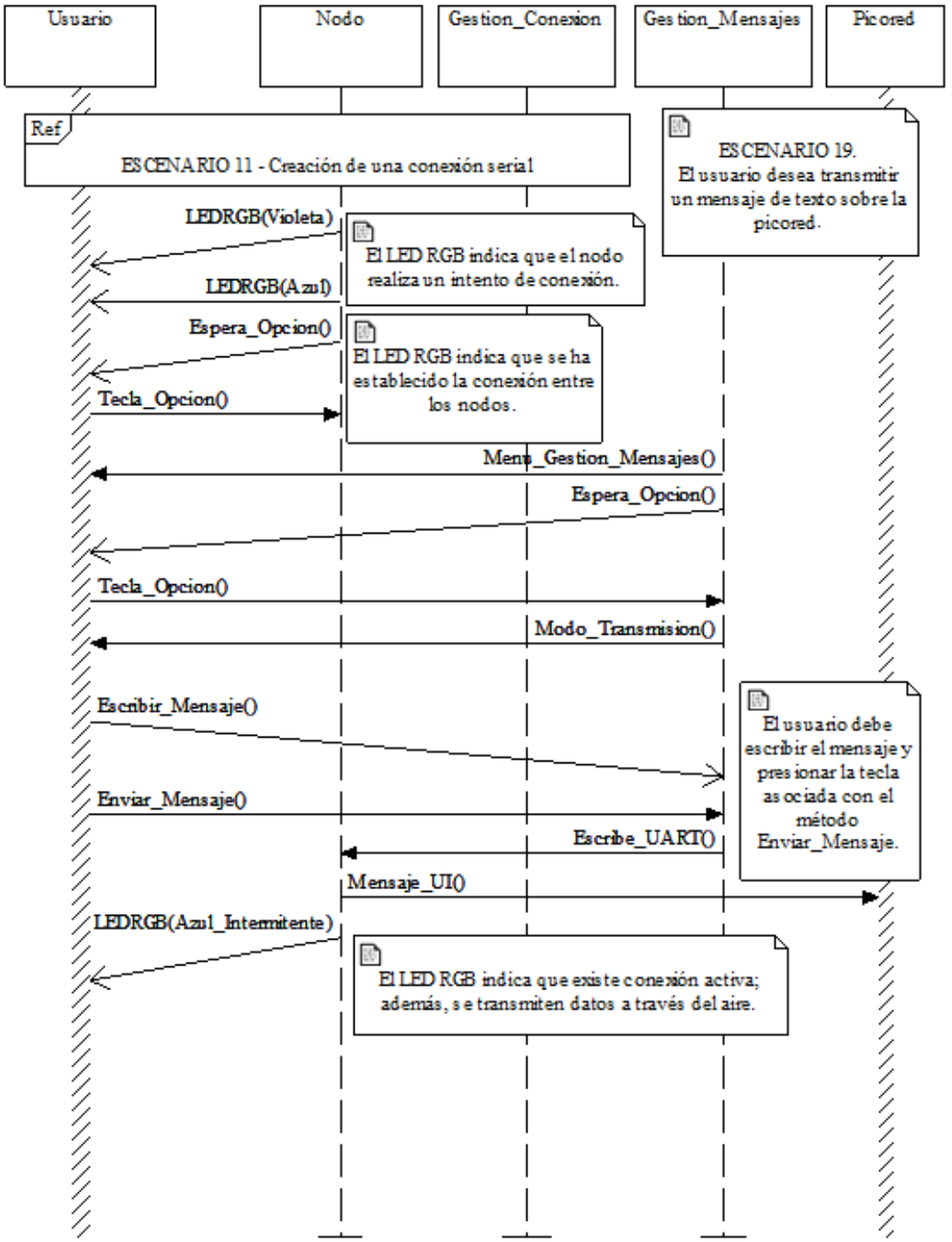


Figura 6.41. Diagrama de secuencia para el Escenario 19: Transmisión un mensaje de texto sobre la picored

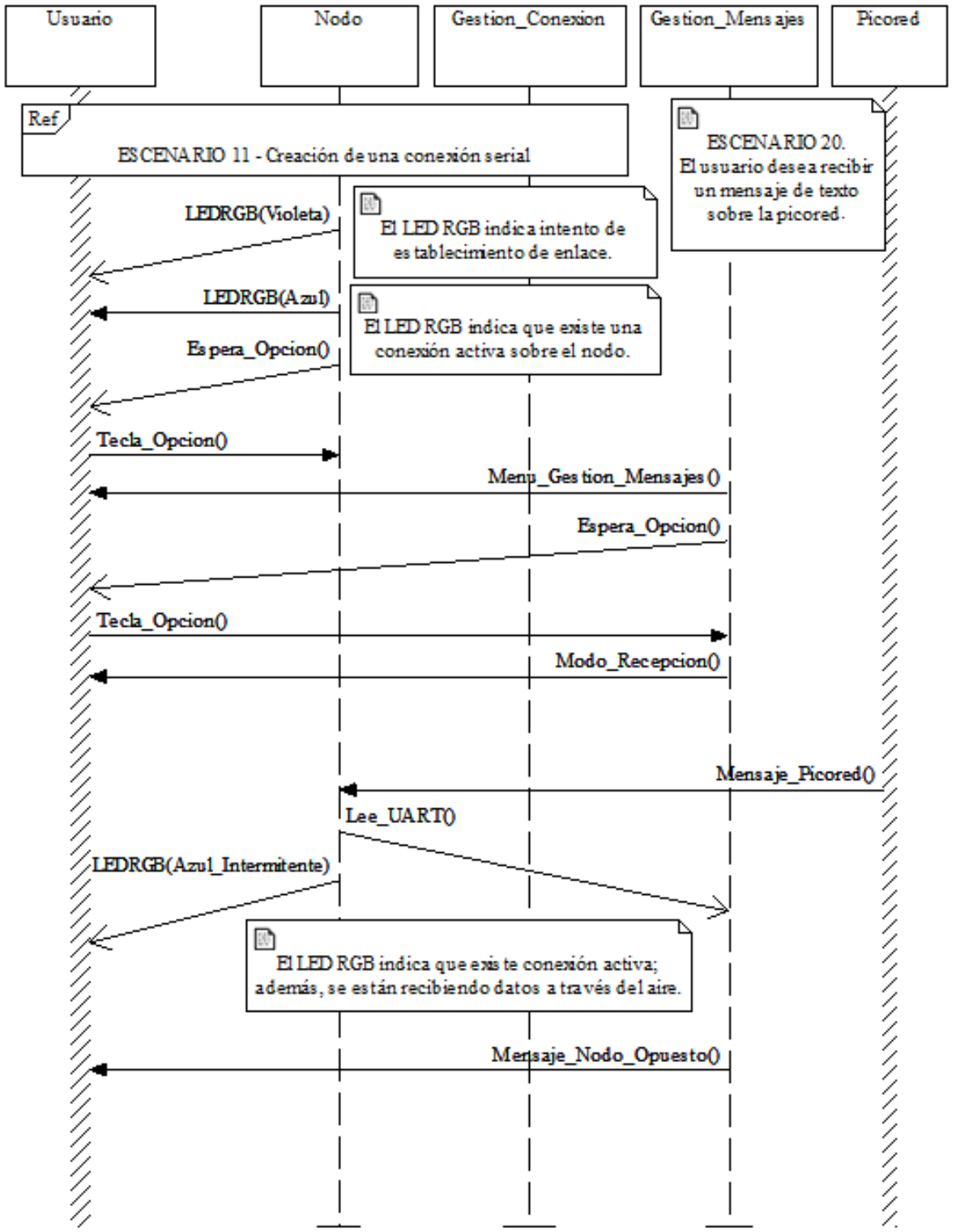


Figura 6.42. Diagrama de secuencia para el Escenario 20: recpción de un mensaje de texto sobre la picored

A continuación se presentan los diagramas de máquinas de estados realizados para la tesis. En los diagramas observamos que en cada instante de tiempo la máquina se encuentra en un estado, y dependiendo de las entradas, actuales y pasadas, que provienen del ambiente, la máquina cambia, o no, de estado pudiendo realizar acciones que a su vez influyen en el ambiente. Así, se presenta en las Figuras 6.43, 6.44, 6.45, 6.46, 6.47, las desiciones que el usuario debe realiazar.

Los diagramas de estado son particularmente útiles para modelar SE porque estos suelen ser reactivos. Por otra parte existen plug-in para poder incluir statecharts en los modelos que se diseñan con Simulink.

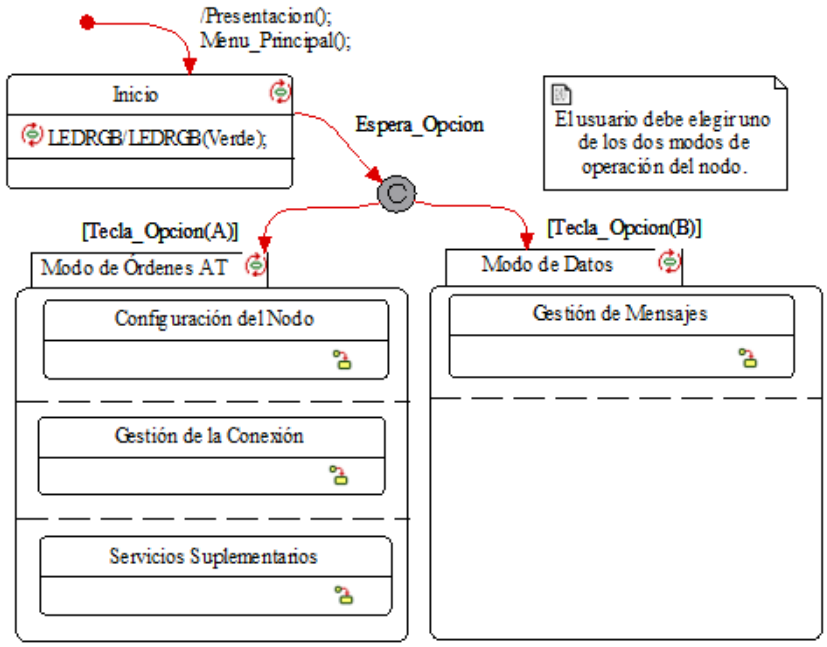


Figura 6.43. Máquina de estados para elegir modo de operación

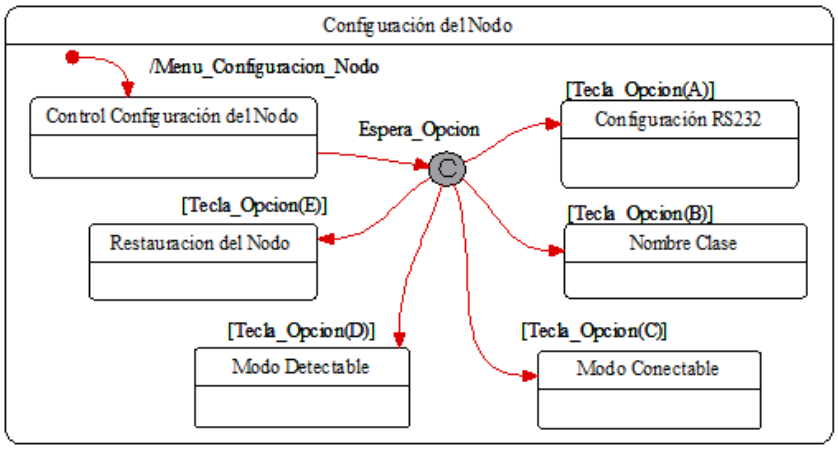


Figura 6.44.Máquina de estados dentro del subsistema Configuración del Nodo

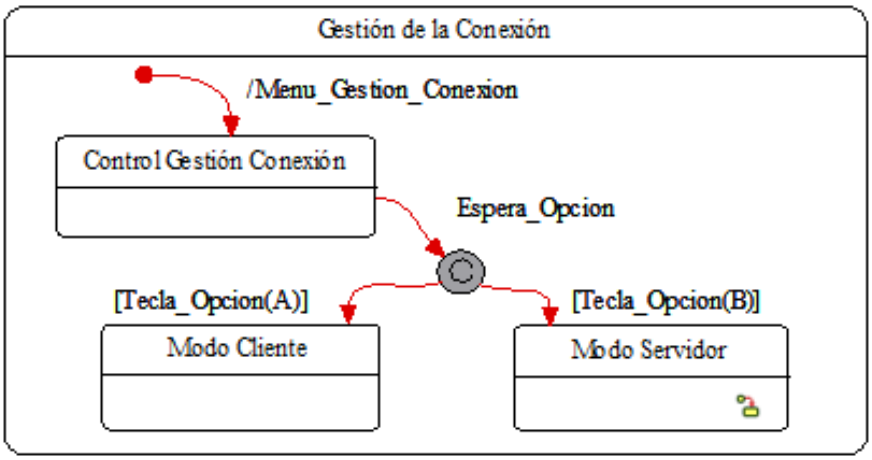


Figura 6.45. Máquina de estados para el establecimiento de una conexión serial

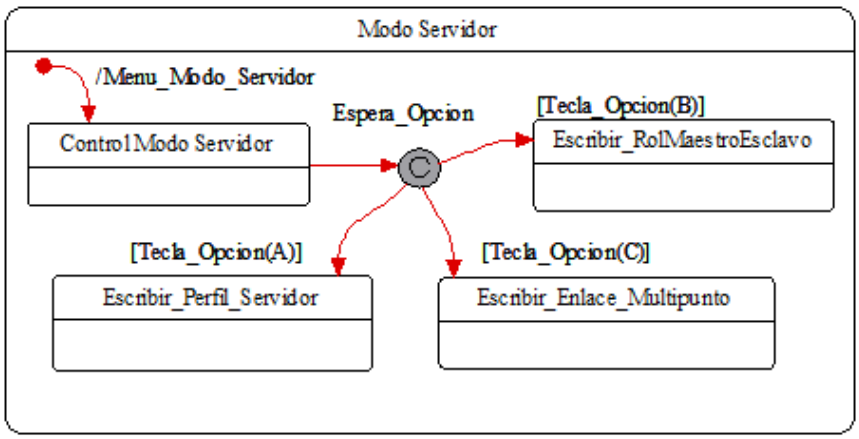


Figura 6.46. Máquina de estados dentro de la generalización Modo Servidor

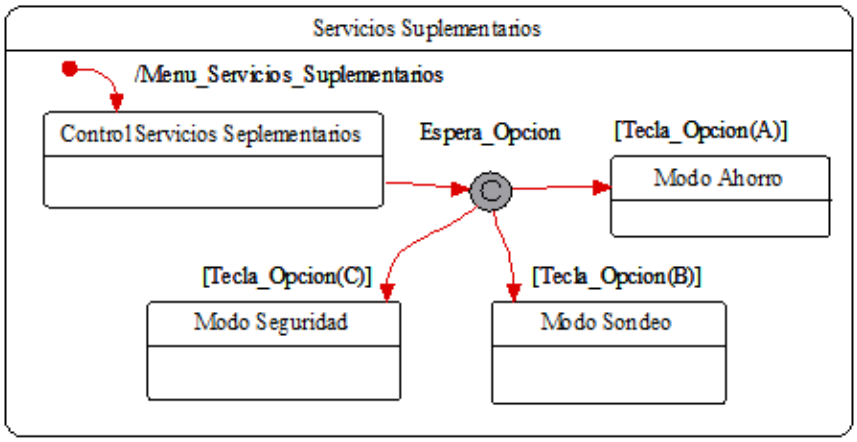


Figura 6.46. Máquina de estados dentro de los servicios suplementarios

7

CONCLUSIONES Y TRABAJO FUTURO

El objetivo de la investigación de esta tesis fue: “Desarrollar una Metodología para el Diseño de Sistemas Empotrados bajo el paradigma de Mejora del Proceso Software”. Así como también, mantener la conformidad con las normas existentes para avanzar en la estandarización de SE; y agilizar los procesos de desarrollo de SE. La metodología SPIES se describió en el capítulo cinco, en el capítulo seis se presentaron los casos de estudio con los que se ha probado la metodología y en el presentante capítulo se enunciarán las conclusiones del trabajo de tesis y el trabajo futuro.

7.1. CONCLUSIONES

Software Process Improvement for Embedded Systems es una metodología diseñada para mejorar el proceso de desarrollo de Sistemas Empotrados y cumple con las características resumidas en la Tabla 7.1

Tabla 7.1. Características de SPIES

Caracteristicas	SPIES
Ámbito de aplicación	Sistemas Empotrados
Artefactos	Si
División de componentes	Si
Modelado	Si
Objetivo	Mejora del proceso de desarrollo de los Sistemas Empotrados
Puntos a favor	Se basa en CMMI y TSP. Propone herramientas para el diseño y desarrollo del SE. Separa el sistema en componentes y propone la reutilización de componentes.
Puntos en contra	Se relaciona con el conocimiento previo de modelos de calidad; lo cual podría dificultar su uso.
Repositorio de componentes	Si
Reutilización de componentes	Si
Separación en fases	Si

El desarrollo de software en SE es vital pues proporciona gran parte del valor del producto, por lo que resulta necesario definir, gestionar, optimizar e institucionalizar el proceso de desarrollo de SE mediante la Mejora del Proceso Software (SPI). Como se mostró a lo largo del documento una de las principales metas de SPI es producir software de calidad, en tiempo, dentro del presupuesto y con la funcionalidad requerida; muy acorde con los retos del desarrollo de SE. Como consecuencia se pensó en una metodología que se basa en los siguientes puntos:

- La metodología guiará a los desarrolladores durante todo el ciclo de vida de los SE a través de la definición de las fases de los procesos, actividades y subactividades.
- Usando CMMI-DEV v1.2 como modelo de referencia, SPIES sigue el paradigma SPI; que planifica, administra y controla el desarrollo de éstos sistemas.
- SPIES, utilizando el modelo TSP para indicar qué usar, cómo usarlo y cuándo usarlo, con un énfasis en la mejora continua. De tal forma que el desarrollo de SE es un proceso planificado, administrado y controlado.

- SPIES utiliza el modelo de calidad (CMMI-DEV v1.2 y TSP) en el desarrollo de SE, mejoraron el ciclo de desarrollo al proporcionar una gestión flexible de los SE.

Los casos de prueba de la metodología presentados en el Capítulo 6 permitieron alcanzar hallazgos importantes:

7.1.1. VENTAJAS DE USAR SPIES

- SPIES está estructurada por ocho fases (desde la Fase 1 hasta la Fase 8), 16 áreas de proceso y cerca de 12 plantillas.
- La capa de gestión se compone por dos áreas de proceso que proporcionan los procesos necesarios para controlar el desarrollo de todo el proyecto. Esta capa se inicia con la realización de un plan de proyecto y termina con la recolección de las lecciones aprendidas en la etapa final. Las actividades que componen esta capa son importantes pero puede ser adaptadas a las necesidades del SE que se esté desarrollando.
- SPIES tiene como tarea desarrollar la arquitectura del sistema y en este proceso divide la complejidad del sistema a nivel subsistema para desarrollaron un SE correcto.
- Mediante la capa de ingeniería, se pueden evitar posibles dependencias de software con hardware.
- SPIES inicia con la planeación por lo que, se deben seguir prácticas efectivas para crear un plan de proyecto, seguir el plan contra la realidad, gestionar los contratos y entregar un proyecto dentro del tiempo y presupuesto establecidos. Para esto, es necesario que los desarrolladores realicen el artefacto "Plan de proyecto" para después iniciar con el diseño del SE, sin embargo, en los casos de prueba se observó que esta fase puede omitirse e iniciar con la especificación de requisitos. En base a esto se afirma que aunque los proyectos que no cuenten con una planeación rigurosa del tiempo pueden reducir este elemento al tener bien identificados los requisitos y componentes que integrarán al sistema.
- El desarrollo y la administración de los requisitos de un SE son tareas vitales para lograr un producto funcional. Los requisitos de los SE van más allá de los requisitos que se obtienen para desarrollar software, pues, un SE tienen asociados requisitos de: software, hardware, diseño, mecánica, entre otros. Por lo tanto es necesario contar con un lenguajes de modelado para describir los requisitos, tal como: UML o el Lenguaje de Modelado de Sistemas (SysML).
- Es necesario contar con un proceso de Diseño del producto que proporcione las actividades para crear un diseño eficaz del sistema basado en los requisitos que se obtienen en la fase de Especificación de requisitos. SPIES indica a los desarrolladores cómo definir la arquitectura de sistema en base a los requisitos y al diseño funcional, lo que ayuda a los desarrolladores a tener una visión clara del producto final.
- La responsabilidad del estudiante por recoger información y retroalimentarse con la experiencia juega un papel importante en esta fase. Toda la información recogida se actualiza en el repositorio de conocimientos para incrementar el conjunto de prácticas efectivas y aplicarlas en proyectos futuros.

7.1.2. DESVENTAJAS DE USAR SPIES

Como resultado del trabajo de investigación se tiene lo siguiente:

- Los métodos, herramientas y técnicas usadas para el desarrollo de SE no sólo varían entre compañías e institutos de investigación, sino también dentro de las mismas empresas y universidades.
- Se observó que los equipos presentaban cierta dificultad al utilizar SPIES por primera vez pues para aplicar la metodología se debe tener un conocimiento previo sobre modelos de calidad. En particular era difícil seguir instrucciones precisas por los equipos de desarrolladores de SE.
- Los equipos que no dominaban el lenguaje UML y siguieron la metodología SPIES externaron que su uso es complicado.

7.2. TRABAJO FUTURO

Como trabajo futuro se plantea refinar el conjunto de prácticas efectivas mediante la experimentación dando un estilo formal de la arquitectura para la construcción de un software automatizado que implemente las prácticas reflejadas en SPIES.

Además de definir y detallar las aéreas de procesos y plantillas que integrarán las siguientes fases:

- Validación. En esta fase se incluirán pruebas antes de que el sistema pase a la fase de entrega y mantenimiento.
- Entrega y mantenimiento. La mayoría de los diseñadores de SE (alrededor del 60%) mantiene y actualiza los productos existentes, en lugar de diseñar nuevos productos. Los desarrolladores deben utilizar la documentación existente y entender el diseño original lo suficiente como para mantener y mejorar. Este proceso requiere de herramientas que se adapten a la reingeniería y establezcan el escenario actual del SE.

Por otra parte se propone trabajar en las siguientes actividades:

- El trabajo presentado específica el proceso a seguir para desarrollar Sistemas Empotrados de calidad, centrado en la mejora del proceso software, sin embargo, es importante considerar en mayor detalle la parte hardware. Por lo que se propone trabajar con los lenguajes de descripción SystemC, VHDL y Rhapsody.
- SPIES establece un conjunto de actividades y tareas a través de la herramienta Rhapsody de Telelogic, Simulink de Matlab, entre otras. Aunque las actividades pueden ser seguidas independientemente de la herramienta de modelado que el desarrollador escoja, como trabajo futuro se planea describir las tareas y actividades para el desarrollo de SE usando herramientas de software y hardware “no privativos”. En consecuencia, se propone trabajar con Fedora ELECTRONIC LAB dedicado al soporte de la innovación y desarrollo traído por la comunidad de Automatización del Diseño Electrónica (EDA) de código abierto. La principal ventaja que ofrece Fedora en su spin electrónica es la construcción de un puente entre las comunidades de software y hardware libres y de código abierto. Lo que podría ayudar a reducir costos de implementación.

BIBLIOGRAFÍA

1. Abrahamsson, P. "Keynote: Mobile Software Development – The Business Opportunity of Today" *Proc. of the International Conference on Software Development* (SWDC-REK), Reykjavik Iceland, pp. 20-23, 2005.
2. Anacleto, V. *El Rol de la Arquitectura de Software en las Metodologías Ágiles. Lineamientos para su Implementación*. Epidata Consulting S.R.L.- Buenos Aires, Argentina. 2005.
3. Arilla, C. & Arribas, L. "Tendencias y Aplicaciones de los Sistemas Embebidos en España" *Boletín* OPTI, No. 39, Segundo Trimestre 2010.
4. Balarin, F., Chiodo, M., Giusto, P., Hsieh, H. & Jurecska, A. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. The Springer International Series in Engineering and Computer Science. 1st Edition, Springer. 1997.
5. Ball, S. *Embedded Microprocessor Systems: Real World Design*. 3rd Edition, Newnes. 2002.
6. Barr, M. *Programming Embedded Systems in C and C++*. 1st Edition, O'Reilly Media. 1999.
7. Basili, V. R. "Quantitative Evaluation of Software Engineering Methodology" Technical Report TR-1519, Department of Computer Science, University of Maryland, College Park, 1985.
8. Baskerville, R., Pries-Heje, J. & Ramesh, B. "The Enduring Contradictions of New Software Development Approaches: A Response to 'Persistent Problems and Practices in ISD'" *Information Systems Journal*, 47(3): 241-245, 2007.
9. Berger, A. *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*.1st Edition, CMP Books. 2002.
10. Birk, A., Järvinen, J., Komi-Sirviö, S., Kuvaja, P., Oivo, M. & Pfahl, D. "PROFES. A Product Driven Process Improvement Methodology" *Proc. of the 1998 European Conference on Software Process Improvement*, Fraunhofer Publica, pp. 78-85, 1998.
11. Booch, G., Rumbaugh, J. & Jacobson, I. *El Lenguaje Unificado de Modelado (Spanish Edition)*. 1ª Edición, Pearson Addison Wesley. 2004.
12. Brey, B. T*he Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processor Architecture, Programming, and Inter- facing*. 6th Edition, Prentice Hall. 2002.
13. Cabo, J. M. & Moralejo, R. "Desarrollo de Instrumentos de Evaluación Educativa para Tecnologías Específicas desde la Perspectiva de Ciencia-Tecnología y Sociedad" *Proyecto Leonardo*, 3(1): 37-52, 2008.
14. Calero, C., Moraga, M. A. & Piattini, M. *Calidad del Producto y Proceso Software*. 1a Edición, Ra-Ma Editorial. 2010.
15. Capote, J., LLantén, J., Pardo, C. & Collazos, C. "Gestión del conocimiento en un programa de mejora de procesos de software en MiPyMEs: modelo KMSPI". *Revista de la Facultad de Ingeniería - Universidad de Antioquia*, (50): 205-216, 2009.
16. Chrissis, M., Konrad, M. & Shrum, S. CMMI. *Guía para la Integración de Procesos y Mejora de Productos (Spanish Edition)*. 2nd Edition, Pearson Addison Wesley. 2009.
17. CMMI Product Team. *CMMI for Development (CMMI-DEV, V1.2)*. CMU/SEI-2006 TR-008, Software Engineering Institute, Carnegie Mellon University. 2006.
18. CMMI Product Team. *CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMISE/SW/IPPD/SS, V1.1). Continuous Representation*. CMU/SEI-2002-TR-011, Software Engineering Institute, Carnegie Mellon University. 2002.

19. Conradi, R., Dybå, T., Sjøberg, D. I. K. & Ulsund, T. "Lessons Learned and Recommendations from Two Large Norwegian SPI Programmes" Proc. of the 9th International Workshop in Software Process Technology (EWSTP 2003), LNCS 2768, pp. 32-45, 2003.
20. Craft, B. *First Steps with Embedded Systems*. Tech Paper. 2002.
21. Deming, W. *Out of the Crisis*. 1st Edition, MIT Center for Advanced Engineering Study. 1986.
22. Edwards, S., Lavagno, L., Lee, E. A. & Sangiovanni-Vincentelli, A. "Design of Embedded Systems: Formal Models, Validation, and Synthesis" *Proceedings of the IEEE*, 85(3): 366-390, 1997.
23. Feng, Q. "Six Sigma: Continuous Improvement Toward Excellence" *Collaborative Engineering: Theory and Practice*. Springer Science + Business Media, LLC 2008, pp. 43-60, 2008.
24. Galeano, G. *Programación de Sistemas Embebidos en C*. 1ª Edición, Alfaomega Grupo Editor. 2009.
25. Garzás, J., Fernández, C. & Piatinni, M. "Una Aplicación de ISO/IEC 15504 para la Evaluación por Niveles de Madurez de PYMEs y Pequeños Equipos de Desarrollo" *Revista Española de Innovación, Calidad e Ingeniería del Software*, 5(2): 88-98, 2009.
26. Genßler, T., Christoph, A., Winter, M., Nierstrasz, O., Ducasse, S., Wuyts, R., Arévalo, G., Schönhage, B., Müller, P. & Stich, C. "Components for Embedded Software: The PECOS Approach" *Proc. of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (CASES 2002), ACM Publisher, pp. 19-26, 2002.
27. Gibson, D. Kost, K. & Goldenson, K. "Performance Results of CMMI-Based Process Improvement". CMU/SEI-2006-TR-004. Software Engineering Institute. Carnegie Mellon. 2006.
28. González, P. & Urrego, G. "Modelo de Requisitos para Sistemas Embebidos" *Revista de Ingenierías Universidad de Medellín*, 7(13): 111-127, 2008.
29. Graaf, B., Lormans, M. & Toetenel, H. "Embedded Software Engineering: The State of the Practice" *IEEE Software*, 20(6): 61-69, 2003.
30. Graaf, B., Lormans, M. & Toetenel, H. "Software Technologies for Embedded Systems: An Industry Inventory", *Proc. of the 4th International Conference on Product Focused Software Process Improvement* (PROFES 2002), LNCS 2559, pp. 453-465, 2002.
31. Greene, B. "Agile methods applied to embedded firmware development" *Proc. of the Agile Development Conference* (ADC'04), IEEE Computer Society, pp. 71-77, 2004.
32. Guler, I., Guillén, M. & Macpherson, J. "Global competition, institutions, and the diffusion of organizational practices: the international spread of ISO 9000 quality certificates" Administrative Science Quarterly, 47(2002): 207-232, 2002.
33. Humphrey, W. *Introduction to the Personal Software Process*. SEI Series in Software Engineering. 1st Edition, Addison-Wesley, Reading, MA. 1997.
34. Humphrey, W. *Introduction to the Team Software Process*. SEI Series in Software Engineering. 1st Edition, Addison-Wesley. 2000.
35. Ibrahim, A., Zhao, L. & Kinghorn, J. "Embedded Systems Development: Quest for Productivity and Reliability" *Proc. of the Fifth International Conference on Commercial-Off-The-Shelf (COTS)-Based Software Systems* (ICCBSS 2006), IEEE Computer Society, pp. 23-32, 2006.
36. ISO/IEC 12207: 2008, Systems and Software Engineering – Software Life Cycle Processes, Geneva, 2008.
37. ISO/IEC 15504-2:2003/Cor. 1:2004 (E): Information Technology – Process Assessment – Part 2: Performing an Assessment. International Organization for Standardization: Geneva. 2004.

38. Jalote, P. *CMM in Practice: Processes for Executing Software Projects at Infosys*. 1st Edition, Addison-Wesley Professional. 2000.
39. Jarvinen, J., Hamann, D., & Van Solingen, R. "On integrating assessment and measurement: towards continuous assessment of software engineering processes" *Proc. of the 6th International Software Metrics Symposium, IEEE Computer Society*, pp. 22-30, 1999.
40. Jonassen, A. M. *Configuration Management Principles and Practice*. 1st Edition, Addison-Wesley Professional. 2003.
41. Jun, D., Rui, L. & Yi-min, H. "Software Process Improvement and Specifications for Embedded Systems". *Proc. of the 5th ACIS International Conference on Software Engineering Research, Management and Applications* (SERA 2007), IEEE Computer Society, pp. 13-18, 2007.
42. Kang, B., Kwon, Y. & Lee, R. "A Design and Test Technique for Embedded Software", *Proc. of the Third ACIS International Conference on Software Engineering Research, Management and Applications* (SERA 2005), IEEE Computer Society, pp. 160-165, 2005.
43. Komi-Sirviö, S. *Development and Evaluation of software process improvement methods*. VTT Publications, Vol. 535, VTT Research Center of Finland, 2004.
44. Kulpa, M. & Johnson, K. *Interpreting the CMMI: A Process Improvement Approach*. 2nd Edition, Auerbach Publications. 2008.
45. Kuvaja, P., Koch, P., Mila, L., Krzanik, A., Bicego, S. & Saukkonen, G. *Software Process Assessment and Improvement: BOOTSTRAP Approach*. 1st Edition, Blackwell Publishers. 1994.
46. Larman, C. *UML y Patrones (Spanish Edition)*. 2nda Edición, Prentice Hall. 2004.
47. Li, C., Zhou, X., Dong, Y. & Yu, Z. "A Formal Model for Component-Based Embedded Software Development" *Proc. of the International Conference on Embedded Software and Systems* (ICESS'09), IEEE Computer Society, pp. 19-23, 2009.
48. Li, Q. & Yao, C. *Real-Time Concepts for Embedded Systems*. 1st Edition, CMP Books. 2003.
49. Liggesmeyer, P. & Trapp, M. "Trends in Embedded Software Engineering" *IEEE Software*, 26(3): 19-25, 2009.
50. Linderman, K., Schroeder, R., Zaheer, S. & Choo, A. S. "Six Sigma: A Goal-Theoretic Perspective" *Journal of Operations Management*, 21(2): 193-203, 2003.
51. Lowenthal, J. *Guía para la Aplicación de un Proyecto Seis Sigma*. 1a Edición, Fundación Confemetal. 2002.
52. Marwedel, P. *Embedded System Design*. 1st Edition, Springer. 2006.
53. McFeeley, B. "IDEAL: A User's Guide for Software Process Improvement" *CMU/SEI-96-HB-001*, Software Engineering Institute, Carnegie Mellon University. 1996.
54. Miranda, L. N. *Seis Sigma/ Six Sigma: Guia Para Principiantes/ Guide for Beginners*. 1a Edición, Panorama México Editorial. 2006.
    Mutafelija, B. & Stromberg, H. "Exploring CMMI-ISO 9001:2000 Synergy when Developing a Process Improvement Strategy" *Proc. of the Software Engineering Process Group 2003 Conference*, pp. 65-86, 2003.
55. Noergaard, T. *Embedded Systems Architecture. A Comprehensive Guide for Engineers and Programmers*. 1st Edition, Newnes. 2005.
56. Paulk, M., Weber, V., Garcia, S., Chrissis, M. & Bush, M. "Key Practices of the Capability Maturity Model (Version 1.1)" *Technical Report CMU/SEI-93-TR-25*, Software Engineering Institute, Pittsburgh, PA, 1993.
57. Paulk, M. "Using the Software CMM with Good Judgment" *ASQ Software Quality Professional*, 1(3): 19-29, 1999.

58. Pearson, C., Giuma, T. & Harris, A. "Transparent Connectivity for Embedded System Design", *Proc. of the Third International Multi-Conference on Computing in the Global Information Technology* (ICCGI '08), IEEE Computer Society, pp. 68-74, 2008.
59. Persse, J. *Process Improvement Essentials: CMMI, Six Sigma, and ISO 9001*. 1st Edition, O'Reilly Editorial. 2006.
60. Pino, F., Garcia, F. & Piattini, M. "Software process improvement in small and medium software enterprises: a systematic review" *Software Quality Journal*, 16(2): 237-261, 2007.
61. Podeswa, H. UML for the IT Business Analyst. 2nd Edition, Course Technology PTR. 2009.
Powel, B. Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. 1st Edition, Addison-Wesley Professional. 1999.
62. Powel, B. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. 1st Edition, Addison-Wesley Professional. 1999.
63. Powel, B. *Real Time UML: Advances in the UML for Real-Time Systems*. 3rd Edition, Addison-Wesley Professional. 2004.
64. Powel, B. *Real-Time UML Workshop for Embedded Systems*. Embedded Technology Series. 1st Edition, Newnes. 2006.
65. Riccobene, E., Scandurra, P., Rosti, A. & Bocchio, S. "A Model-driven Design Environment for Embedded Systems" *Proc. of the 43rd Annual Design Automation Conference* (DAC'06), ACM Publisher, pp. 915-918, 2006.
66. Salgado, J. "Sistemas embebidos, soluciones optimizadas" Revista Dialnet. ISSN 0300-3787, Nº 394, 2008 , págs. 58-63.
67. Sangiovanni-Vincentelli, A. & Martin, G. "Platform-Based Design and Software Design Methodology for Embedded Systems" *IEEE Design & Test of Computers*, 18(6): 23-33, 2001.
68. Scalone, F. "Estudio comparativo de los Modelos y Estándares de Calidad del Software" *Tesis de Maestría en Ingeniería en Calidad*. Universidad Tecnológica Nacional - Facultad Regional Buenos Aires, 2006.
69. Seceleanu, C., Vulgarakis, A. & Petterson, P. "REMES: A Resource Model for Embedded Systems", *Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems* (ICECCS 2009), IEEE Computer Society, pp. 84-94, 2009.
70. Sentilles, S., Pettersson, A., Nyström, D., Nolte, T., Pettersson, P. & Crnkovic. I. "Save-IDE – A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems" *Proc. of the IEEE 31st International Conference on Software Engineering* (ICSE'09), IEEE Computer Society, pp. 607-610, 2009.
71. Serrano, M., Montes de Oca, C. & Cedillo, K. "An Experience on Implementing the CMMI in a Small Organization Using the Team Software Process" *Proc. of the First International Research Workshop for Process Improvement in Small Settings*, CMU/SEI-2006-SR-001, pp. 5-10, 2006.
72. Shaout, A., El-Mousa, A. H. & Mattar, K. "Models of Computation for Heterogeneous Embedded Systems" *Electronic Engineering and Computing Technology*, Lecture Notes in Electrical Engineering, Volume 60, pp. 201-213, 2010.
73. Solingen, R. "Integrating Software Engineering Technologies for Embedded Systems
74. Development". *Product Focused Software Process Improvement*, Lecture Notes in Computer Science, Vol. 2559, pp. 466-474, 2002.
75. Somerville, I. *Software Engineering*. 9th Edition, Addison Wesley. 2010.

76. Srinivasan, J., Dobrin, R. & Lundqvist, K. "'State of the Art' in Using Agile Methods for Embedded Systems Development" *Proc. of the 33rd Annual IEEE International Computer Software and Applications Conference* (COMPSAC'09), IEEE Computer Society, pp. 522-527, 2009.
77. Stienen, H. Software Process Assessment and Improvement: Five Years of Experiences with BOOTSTRAP. 1st Edition, Wiley-IEEE Computer Society Press. 1999.
78. Sumano, P. "Desarrollo de una Metodología para la Gestión de los Servicios de Software Subcontratado en las Pequeñas Empresas de Software" *Tesis de Maestría en Ingeniería de Software*. Universidad Tecnológica de la Mixteca, 2009.
79. Tocci, R., Widmer, N. & Moss, G. *Digital Systems: Principles and Applications*. 11th Edition, Prentice Hall. 2010.
80. Vahid, F. & Givargis, T. *Embedded System Design: A Unified Hardware/Software Introduction*. 1st Edition, Wiley. 2002.
81. Zapata, M. & Garcés, L. "Generación del Diagrama de Secuencias de UML 2.1.1 desde Esquemas Pre Conceptuales" *Revista EIA*, (10): 89-103, 2008.

URLS

[URL-1] *Embedded Computing Design. Identifying Best Practices for Developing Embedded Systems*. http://embedded-computing.com/identifying-best-practices-for-developing-embedded-systems. Última consulta: Mayo, 2011.

[URL-2] EE|Times Design. *An Insider's View of the 2008 Embedded Market Study*. http://www.eetimes.com/design/embedded/4007664/An-insider-s-view-of-the-2008-Embedded-Market-Study. Última consulta: Mayo, 2011.

[URL-3] AENOR. *Calidad y Madurez en el Desarrollo del Software*. http://www.iso15504.es/attachments/110_une242.pdf. Última consulta: Mayo, 2011.

[URL-4] SCD Source. *Ten 2008 Trends in System and Chip Design*. http://www.edxact.com/PDF2008/080102_citation_SCDSOURCE.pdf. Última consulta: Mayo, 2011.

APÉNDICE A
GLOSARIO DE TÉRMINOS

Alcance del proyecto: es la definición de los límites y la suma total de todos los productos y requisitos o características que tendrá el sistema empotrado. Consiste en un proceso de subdividir los entregables principales en componentes administrables, para cumplir con los objetivos.

Análisis de requisitos: es la determinación del rendimiento y características funcionales específicas del producto; se encuentran los intereses que él SE debe cubrir. Responde a la pregunta ¿Qué es lo que debe hacer el sistema empotrado? Involucra un trabajo técnico de grupo con los clientes/ usuario, para averiguar el dominio de la aplicación, los servicios que el sistema debe proporcionar y las restricciones operacionales del sistema.

Análisis de riesgos: es la evaluación, clasificación y priorización de los riesgos que pueden existir en el desarrollo de un SE.

Arquitectura del proceso: es el diseño estructural de sistemas de proceso generales. La arquitectura de proceso describe: interfaces, interrelaciones y otras relaciones entre los elementos de proceso.

Arquitectura del producto: La arquitectura de producto siempre tiene una gran importancia, se busca diseñar una arquitectura teniendo en cuenta los diferentes componentes del proyecto, lo que permite obtener un producto escalable y fácil de usar e implementar. Estas prácticas permiten la reutilización de códigos lo que puede disminuir los tiempos de desarrollo y costos finales.

Arquitectura del sistema: es el conjunto de decisiones significativas acerca de la organización del SE y se basa en la especificación del conjunto de subsistemas, la ubicación de los requisitos, la funcionalidad de los subsistemas y las interfaces entre ellos.

Arquitectura funcional: La organización jerárquica de las funciones, de sus interfaces funcionales internos y externos e interfaces físicas externos, de sus respectivos requisitos funcionales y de rendimiento, y de sus restricciones de diseño.

Arquitectura: es una descripción de la organización, motivación y estructura de un sistema empotrado. Está definida en términos de componentes y sus conexiones. Implica diferentes tipos de arquitectura, por ejemplo: la arquitectura de hardware, la arquitectura software, etc. La estructuración es básica y permite asignar tareas en el equipo de desarrollo. Una arquitectura consiste de varias vistas que cubren diferentes aspectos del sistema en general.

Artefacto: un artefacto es una pieza de información que es utilizada o producida por un proceso de desarrollo software. Es un producto tangible como por ejemplo: caso de uso, diagramas de clases, planes de proyecto, etc.

Caja límite: es un bloque de construcción que delimita las fronteras del sistema mediante una caja. Los elementos que queden fuera de la caja forman parte del entorno; los actores están fuera de la caja límite.

Calendario del proyecto: calendario base utilizado por un proyecto. Refleja la totalidad de los días y horas laborables del proyecto, así como los períodos no laborables.

Calendario limitado por recursos: gestión del tiempo destinado al desarrollo del proyecto en base al conjunto de elementos disponibles para desarrollar el SE.

Calendario orientado por eventos: gestión del tiempo destinado al desarrollo del proyecto en base a sucesos importantes y programados.

Calidad: Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor.

COCOMO: Modelo Constructivo de Costes, es un modelo matemático de base empírica utilizado para estimación de costes de software.

Componente: es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la realización de dicho conjunto.

Contrato: conjunto de todas las condiciones relacionadas con una interfaz. Especifica las entradas y salidas permitidas en un componente.

Datos históricos: son los antecedentes necesarios para llegar al conocimiento exacto para deducir consecuencias. En base a datos históricos se estima el tiempo, recursos y costos del desarrollo de un SE.

Definición de requisitos: descripción orientada al cliente de las funciones del sistema y las restricciones en su operación. Incluye a los requisitos funcionales y no funcionales.

Dependencias del calendario: en general las actividades del proyecto dependen de otras; obedeciendo así el comienzo o del final de otra, para poder comenzar o finalizar.

Desarrollo de sub sistemas: se refiere a la producción del conjunto de elementos que componen al SE.

Diseño de componentes: es la descripción de la descomposición de los subsistemas en componentes.

Diseño de interfaces: proceso que utiliza los productos del análisis para producir una colección de operaciones que se utilizan para especificar un componente. Describiendo las interfaces de los subsistemas.

Diseño de la arquitectura: es el proceso de diseño inicial que identifica los subsistemas y establece un marco de trabajo para el control y comunicación de los subsistemas.

Diseño del sistema: se ocupa de desarrollar las directrices propuestas durante el análisis en función de aquella configuración que tenga más posibilidades de satisfacer los objetivos planteados tanto desde el punto de vista funcional como del no funcional.

Diseño funcional: es el diseño del sistema creado en base a los requisitos del SE. Es una descripción lógica de cómo trabajará el sistema y cubre las funcionalidad de un SE sin considerar los detalles técnicos de la implementación.

Diseño mecánico: consiste en la concepción, descripción o bosquejo de la solución a construir que satisface los requisitos mecánicos del SE.

Diseño software: consiste en la concepción, descripción o bosquejo de la solución a construir que satisface los requisitos de software del SE. Durante este proceso se desarrollan diagramas UML.

Diseño: un proceso que utiliza los productos del análisis para producir una especificación para implementar un sistema. Es un modelo concreto que indica exactamente cómo serán realizadas las propiedades esenciales.

Documento de requisitos: es la declaración oficial de lo que es requerido para que el sistema sea desarrollado. Incluye la definición y especificación de requisitos.

Eficaz: el sistema empotrado desarrollado no debe desperdiciar los recursos del sistema.

Entregable: es cualquier producto medible y verificable que se elabora para completar un proyecto o parte de un proyecto.

Escenario: secuencia especifica de acciones que ilustra un comportamiento.

Especificación de abstracciones: consiste en especificar los subsistemas que integran al sistema empotrado.

Especificación de requisitos: descripción detallada y precisa de la funcionalidad del sistema y de sus restricciones. Escrito en un lenguaje técnico. El propósito es definir las propiedades esenciales del sistema que será desarrollado

Especificación de software: descripción detallada de software, la cual puede servir como una base para el diseño o implementación.

Especificación: establece los requisitos y restricciones del sistema empotrado.

Funcionalidad: lo que hace practico y utilitario al SE. Un producto es funcional cuando su diseño u organización se ha atendido, sobre todo, a la facilidad, utilidad y comodidad de su empleo.

Gestión del tiempo: consiste en la distribución del tiempo en las actividades del proyecto.

Implementación hardware: es la tarea de pasar de un dominio de abstracción a

Implementación: consiste en poner en funcionamiento el SE, convirtiendo la fase de diseño a código y/o hardware.

Ingeniería de software: es la disciplina o área de la ingeniería que ofrece métodos y técnicas para desarrollar y mantener software.

Ingeniería inversa: consiste en obtener información a partir de un producto, con el fin de determinar de qué está hecho, qué lo hace funcionar y cómo fue fabricado.

Integración del sistema: es el proceso de conjuntar el hardware y el software del sistema empotrado. Por lo general el proceso se realiza de forma incremental, de forma que los subsistemas son integrados uno a la vez. La integración de todos los componentes en un mismo sistema debe utilizar mecanismos de comunicación.

LOC: Líneas de código fuente.

Medidas del tiempo: para medir el tiempo en los proyectos de SE, se puede utiliza el sistema sexagesimal; cuyas unidades son: la hora, el minuto y el segundo.

Medidas: cada una de las unidades que se emplean para medir tiempo, costo o recursos empleados para el desarrollo de un SE.

Modelado de la arquitectura del sistema: presenta una visión abstracta de los subsistemas que configuran el sistema. Incluye los números de componentes, su conexión y los tipos de componentes, así como, los flujos de información entre subsistemas e identifica distintos tipos de componentes funcionales del modelo.

Modelo: es la simplificación de la realidad, creada para comprender mejor el sistema que se está creando. En UML, la unidad del modelo es el paquete.

Modelos del sistema: representación abstracta del sistema cuyos requisitos están siendo analizados. El modelado del sistema permite entender la funcionalidad del sistema. Los modelos son utilizados para comunicarse con los clientes.

Plan de proyecto: El plan que proporciona la base para ejecutar y controlar las actividades del proyecto, las cuales tratan los compromisos con el cliente del proyecto. La planificación del proyecto incluye estimar los atributos de los productos de trabajo y de las tareas, determinar los recursos necesarios, negociar los compromisos, elaborar un calendario, e identificar y analizar los riesgos del proyecto.

Plan: describe la forma de cómo será realizado un proyecto especíifico (cuándo, cómo y qué costo).

Planificación: plan general, metódicamente organizado y frecuentemente de gran amplitud, para obtener los objetivos.

Presupuesto del proyecto: la creación de un presupuesto para el proyecto permite al jefe de proyecto especificar la capacidad máxima de gasto de dinero, ejecución de trabajo o uso de materiales de un proyecto.

Producto: artefacto del desarrollo, tal como los modelos, el código, la documentación y los planes de trabajo.
Proyecto: serie de pasos que típicamente va a producir un producto.

Pruebas: es un proceso que consiste en verificar que el sistema empotrado cumple con las especificaciones requeridas.

Repositorio de activos o artefactos: es el lugar donde se almacenan los resultados, materiales, etc producto del desarrollo de un SE.

Requisito operacional: es aquel que especifica cómo colabora el sistema con otros elementos en su entorno.

Requisito: característica, propiedad o comportamiento deseado de un SE. Son instrucciones abstractas de alto nivel de un sistema, limitada a desarrollar una especificación funcional. Permiten establecer lo que el cliente quiere del sistema.

Requisitos de diseño: tiene que ver con las características del diseño mismo pero no del sistema. Requisitos funcionales: describe que debe realizar el sistema empotrado para el usuario. Estos requisitos quedan recogidos en los casos de uso.

Requisitos no funcionales: no van asociados a casos de uso, y son los requisitos resultantes de las restricciones impuestas por el entorno y la tecnología, especificaciones sobre tiempo de respuesta o volumen de información tratado por unidad de tiempo, requisitos en cuanto a interfaces, facilidad de mantenimiento, etc.

Riesgo del proyecto: El riesgo en un proyecto es un evento incierto o condición incierta que si ocurre, tiene un efecto positivo o negativo sobre el proyecto.

Seguimiento del tiempo: es una forma de mejorar la calidad del trabajo.

Tarea: define a un elemento de trabajo.

Sistema: conjunto de componentes interrelacionados trabajando conjuntamente para un fin común. El sistema puede incluir software, hardware y dispositivos mecánicos.

Trabajo: es una actividad que se hace con respecto al proyecto o a una tarea específica.

Trazabilidad de requisitos: la trazabilidad de requisitos es considerada un proceso imprescindible para la adecuada gestión de requisitos. La trazabilidad de requisitos es una asociación discernible entre los requisitos y los requisitos relacionados, las implementaciones y las verificaciones.

TSP (Team Software Process): es una metodología para dirigir el trabajo de mejora y desarrollo de software además de establecer un entorno donde el trabajo efectivo de equipo sea normal y natural. Incorporar TSP acelera el cumplimiento de las prácticas de CMMI de una forma más generalizada en el desarrollo de SE.

Validación de requisitos: demostración de que los requisitos que definen el sistema son los que realmente quiere el cliente.

Validación: responde a la pregunta ¿Estoy haciendo el sistema correcto? Asegura que todos los requisitos del sistema hayan sido cubiertos.

Verificación: es el proceso por el que se garantiza que el sistema diseñado muestra las especificaciones requeridas. Responde a la pregunta ¿Estoy haciendo el sistema de forma correcta?

APÉNDICE B
ACRÓNIMOS

ADL	Descripción de la Arquitectura
ASIC	Circuitos Integrados de Aplicación Específica
CI	Circuito Integrado
CMA	Gestión de la Configuración
CMG	Gestión de Contratos
CMM	*Capability Maturity Model*
CMMI	*Capabality Maturity Model Integration*
CMMI/DEV	*Capability Maturity Model Integration for Development*
CMM-IPD	*Capability Maturity Model for Integrated Product Development*
COCOMO	Modelo Constructivo de Costes
DSP	Procesador Digital de Señales
EMF	*Eclipse Modeling Framework*
EPL	Planeación
ESCM	*Embedded Software Component Model*
GPS	Sistema de Posicionamiento Global
hw	Hardware
IDE	*Integrated Development Environment*
ITEA	*Information Technology for European Advacement*
MAN	Medición y Análisis
MCU	Microcontrolador
MDA	*Model Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model Driven Engineering*
ModES	*ModES Model-Driven Design of Embedded Systems*
MPU	Unidad del Microprocesador
NFPs	Propiedades No Funcionales
PCI	Mejora Continua de Productos
PDE	Desarrollo del Producto
PDM	Entrega y Mantenimiento del Producto
PDS	Diseño del Producto
PIM	Modelo Independiente de Plataforma
PIN	Integración del Producto
PM	Modelo de Plataforma
PPQ	Productos y Procesos de Calidad
PSM	Modelo Específico de la Plataforma
PSP	*Personal Software Process*

QIP	*Quality Improvement Paradigm*
QoS	Calidad del servicio
QVT	Query, Vistas, Transformaciones
REMES	*Resource Modelo For Embedded Systems*
RES	Especificación de Requisitos
ROPES	*Rapid Object-Oriented Process For Embedded Systems*
SaveCCT	*Save-Comp Component Technology*
SAVE-IDE	*SAVE INTEGRATED DEVELOPMENT ENVIRONMENT*
SE	Sistemas Empotrados
SECM	Modelo de Madurez y Capacidad de la Ingeniería de Sistemas
SEI	*Software Engineering Institute*
SPI	*Software Process Improvement*
SPIES	*Software Process Improvement For Embedded Systems*
SPP	*Simplified Paralell Process*
SPU	*Software Producing Unit*
sw	Software
SW-CMM	*Modelo de Madurez y Capacidad para el desarrollo de Software*
TSP	*Team Software Process*
UAV	Vehículos Aéreos no-Tripulados
UML	*Unified Modeling Language*
XP	*eXtreme Programming*
LOC	*Line of Code*

APÉNDICE C

FUNDAMENTOS DE MODELADO CON UML

UML PARA SISTEMAS EMPOTRADOS

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML) es un lenguaje estándar de modelado gráfico que puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucre una gran cantidad de software [Booch, Rumbaugh & Jacobson, 2004] o hardware e incluye aspectos como: procesos de negocio, funciones del sistema, bases de datos, etc. El modelado es una parte central de todas las actividades que conducen a la producción de buen software. El bloque UML incluye tres clases de bloques: elementos, relaciones y diagramas.

C.1. ELEMENTOS DE CONSTRUCCIÓN EN UML

C.1.1. ELEMENTOS

Existen cuatro tipos de elementos UML: elementos estructurales, de comportamiento, de agrupación, y de anotación [Booch, Rumbaugh & Jacobson, 2004].

1. Elementos estructurales: Existen siete tipos de elementos estructurales, que representan cosas que son conceptuales o materiales. A continuación se describe de forma breve cada uno y se presentan gráficamente en la Figura C.1.

- Clase: es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.
- Interfaz: es una colección de operaciones que especifican un servicio de una clase o componente; puede representar el comportamiento completo de una clase o componente o sólo una parte de ese comportamiento.

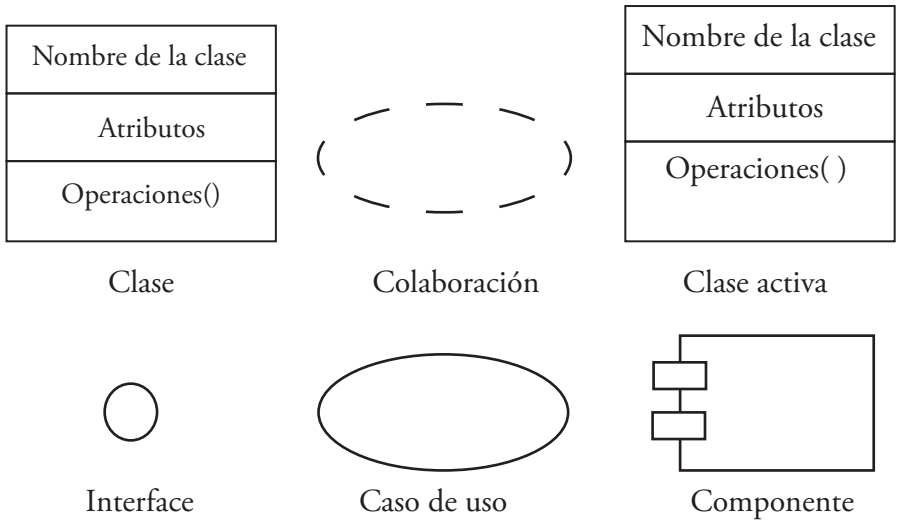


Figura C.1. Elementos estructurales de UML

- Colaboración: define un conjunto de acciones entre objetos para proporcionar cierto comportamiento, y cada objeto juega ciertos roles (tipos) en relación con los demás objetos de la colaboración.
- Caso de uso: describe una interacción entre un actor y el sistema para producir un resultado con un valor.
- Clase activa: es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución y por lo tanto pueden dar origen a actividades de control.
- Componente: es una parte física y reemplazable de un sistema que conforma de un conjunto de interfaces y proporciona la implementación de dicho conjunto.
- Nodo: es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional.

2. Elementos de comportamiento: UML cuenta con dos elementos de comportamiento y son las partes dinámicas de los modelos. A continuación se describen brevemente (véase Figura C.2).

- Interacción: es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico.
- Maquina de estados: es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos.

Figura C.2. Elementos de comportamiento de UML

3. Elementos de agrupación: UML se compone de partes organizativas. Estos son cajas en las que se puede descomponer un modelo. En total, hay un elemento de agrupación:

- Paquete: es un mecanismo de propósito general para organizar elementos en grupos.

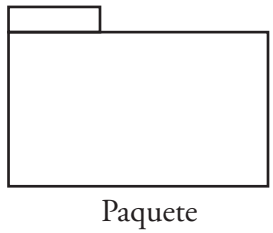


Figura C.3. Elementos de agrupación de UML

4. Elementos de anotación: Los elementos de anotación son las partes explicativas del modelo UML.

- Notas: es un símbolo para mostrar restricciones y comentarios junto a un elemento o colección de elementos.

Nota

Figura C.4. Elementos de anotación de UML

C.1.1.1. RELACIONES

Existen cuatro tipos de relaciones UML y son los bloques básicos de UML para establecer relaciones.

- Dependencia: es una relación semántica entre dos elementos. La relación de dependencia ocurre cuando dos objetos de distinta clase interactúan de forma temporal. Esta relación ocurre típicamente cuando un objeto invoca a otro al interior de uno de sus métodos.
- Asociación: es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos.
- Generalización: es una relación de especialización/generalización en la cual los objetos del elemento especializado pueden sustituir a los objetos del elemento general.
- Realización: es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá.

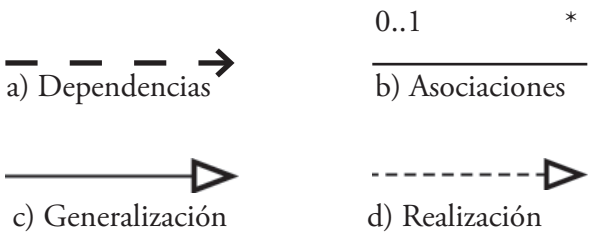


Figura C.5. Relaciones de UML

C.1.1.3. DIAGRAMAS

Los diagramas UML permiten crear vistas de los modelos mentales de un sistema software. Existen cuatro tipos básicos de diagramas UML (véase Figura C.6): diagramas estructurales, diagramas funcionales, diagramas de comportamiento y diagramas de interacción. Los diagramas permiten crear diferentes vistas de un sistema: vista de comportamiento, vista estructural, y vista dinámica. Con estos diagramas resulta más fácil detectar las dependencias y dificultades implícitas del sistema, así como evitar cambios en una etapa final en la cual se incrementa los costos. A continuación se describirán los diagramas UML más importantes para el diseño de un Sistema Empotrado.

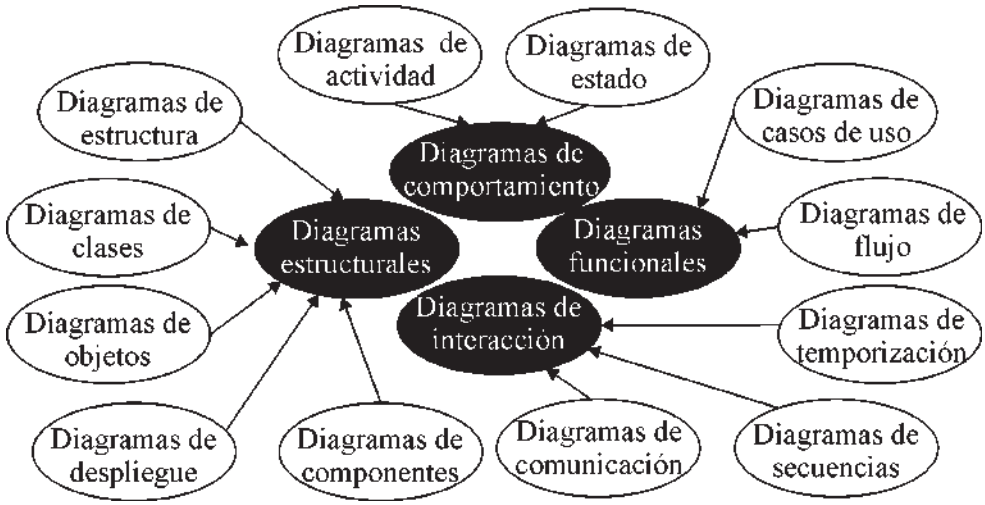


Figura C.6. Tipos de diagramas UML [Powel, 2006]

C.1.1. DIAGRAMA DE CASOS DE USO

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario por [Podeswa, 2009], y es una colección de escenarios con éxito y fallo relacionados, que describen a los actores utilizando un sistema para satisfacer un objetivo [Larman, 2004]. Los casos de uso se utilizan para mejorar la comprensión de los requisitos (especialmente los requisitos funcionales). Es importante recalcar que los casos de uso son documentos de texto, sin embargo, UML define un diagrama de casos de uso para ilustrar los nombres de casos de uso, actores (un tipo especial de clase), y sus relaciones [Larman, 2004]. Son especialmente importantes en el modelado y organización del comportamiento de un sistema [Booch, Rumbaugh & Jacobson, 2004] y permite realizar la especificación del alcance funcional del producto software que se construye y de los actores. El propósito de este diagrama consiste en enumerar a los actores y los casos de uso, mostrando un comportamiento y determinando que actores participan en cada caso [Booch, Rumbaugh & Jacobson, 2004]. El diagrama de casos de uso consta de elementos estructurales y relaciones (véanse Tablas C.1 y C.2).

La representación textual de los casos de uso es una opción y se realiza en base a diferentes grados de formalidad: breve, informal y completo. Existen plantillas disponibles para los casos de uso completos (www.usecases.org). La representación en forma textual de los casos de uso da una descripción de sus componentes, acciones y reacciones. El contenido de la representación textual es el siguiente:

- Nombre del caso de uso
- Actor primario
- Sistema al que pertenece el caso de uso
- Conjunto de actores
- El nivel del caso de uso (objetivo usuario o subsunción)
- Condiciones previas que deben cumplirse para que el caso de uso pueda ser ejecutado
- Operaciones del escenario principal
- Extensiones

Tabla C.1. Elementos estructurales de un caso de uso

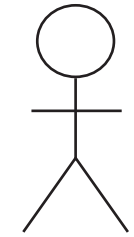
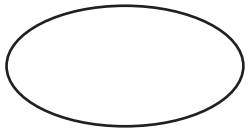
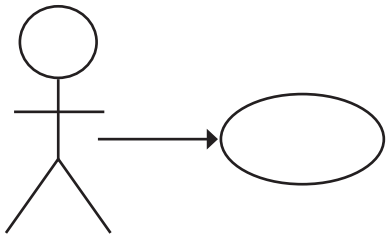
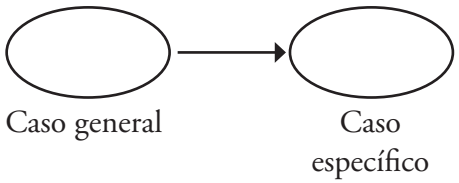
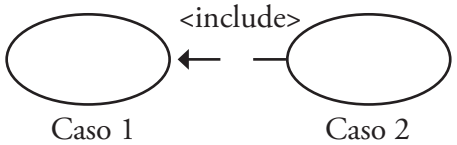
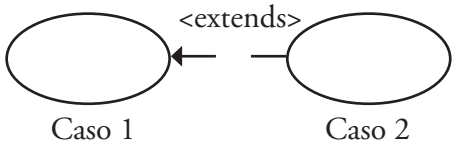
Notación	Descripción
 <<Actor>>	Un *actor* es un conjunto coherente de roles que juegan los usuarios o cualquier otro sistema (organización, software, maquina, etc) de los casos de uso cuando interactúan con éstos [Booch, Rumbaugh & Jacobson, 2004]. Los actores llevan a cabo casos de uso. Un mismo actor puede realizar muchos casos de uso, y un caso de uso puede tener a varios actores. De igual forma los actores pueden servir para identificar los casos de uso que realiza cada uno de ellos. Existen tres tipos de actores [Larman, 2004]: • Actor principal: tiene objetivos de usuario. • Actor de apoyo: proporciona un servicio. • Actor pasivo: está interesado en el comportamiento del sistema). Los actores principales se colocan al lado izquierdo del diagrama.
 <<Caso de uso>>	Un *caso de uso* es la descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecutan un sistema para producir un resultado observable, de valor para el actor [Booch, Rumbaugh & Jacobson, 2004]. Un caso de uso describirá una interacción entre un actor y el sistema, así el caso de uso encapsulará una parte importante del comportamiento del sistema al representar las acciones detalladas que involucran el comportamiento de este.

Tabla C.2. Elementos relacionales de un caso de uso

Notación	Descripción
	La *asociación* es la comunicación que se da entre un actor y un caso de uso, o entre dos casos de uso. Se denota por una línea dirigida entre ellos.
	La *generalización* es el tipo de relación que existe entre un caso de uso general y un caso de uso más específico. Donde este ultimo hereda propiedades del primero y agrega sus propias acciones. La relación de generalización entre los elementos se indica con una flecha grande hueca que señala del elemento más general partiendo del más especializado.
	La *inclusión* es una dependencia que permite identificar comportamientos del sistema comunes a múltiples casos de uso. Se utiliza una dependencia adornada por el estereotipo <<include>>. Permite incorporar el flujo de eventos de un caso de uso pequeño dentro de un caso de uso base de la aplicación. A través de esta relación se re-usa un caso de uso encapsulado en distintos contextos a través de su invocación desde otros casos de uso. La inclusión sirve para compartir una funcionalidad común entre varios caso de uso. También puede emplearse para estructurar un caso de uso describiendo sus subfunciones.
	La relación de extensión amplía la funcionalidad de un caso de uso mediante la extensión de sus secuencias de acciones. Por lo tanto <<extends>> se ocupa cuando se tiene un caso de uso que es similar a otro, pero que hace un poco más.

C.1.2. DIAGRAMA DE CLASES

Este tipo de diagramas muestra un conjunto de clases, interfaces y colaboraciones, [Booch, Rumbaugh & Jacobson, 2004] y sirve para visualizar las relaciones entre las clases que involucran el sistema. Este tipo de diagrama se basa en el uso de clases para crear categorías o grupos de cosas que tienen atributos y acciones similares. En la Tabla C.3 se describen los principales elementos de este tipo de diagramas [Booch, Rumbaugh & Jacobson, 2004]. Los diagramas de clases pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo de partes más grandes. Los diagramas de clase también son la base para los diagramas de: componentes y de despliegue.

Tabla C.3. Elementos del diagrama de clases

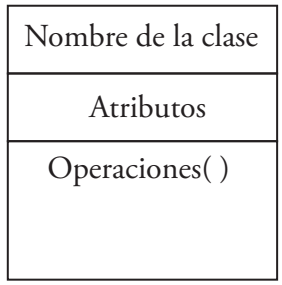
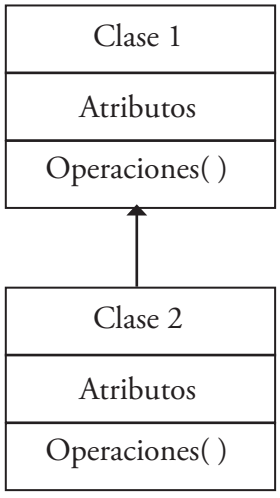
Notación	Descripción
	Una *clase* es el descriptor de un conjunto de objetos que comparten los mismos atributos, métodos, relaciones y comportamiento. Se utilizan para representar elementos del software, conceptuales y/o hardware. Además son útiles en la captura del vocabulario del sistema que se está desarrollando. Las clases bien estructuradas están bien delimitadas y forman parte de una distribución equilibrada de responsabilidades en el sistema. Un *atributo* es una propiedad de una clase identificada con un nombre, que representa alguna propiedad del elemento que se está modelando que es compartida por todos los objetos de esa clase. Una *operación* es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase.
	Una *asociación* es una descripción de un conjunto de enlaces relacionados entre objetos de dos clases. Es así una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. La asociación se considera bidireccional, lo que quiere decir que es posible una conexión lógica entre los objetos de una clase y los de la clase asociada.

Tabla C.3. Elementos del diagrama de clases (continuación)

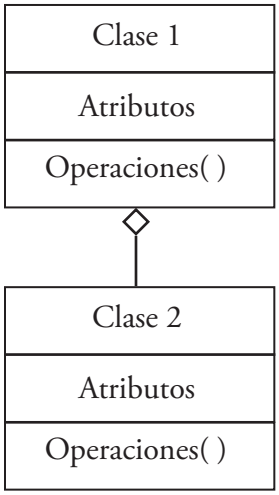
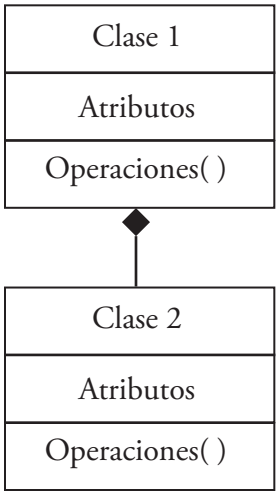
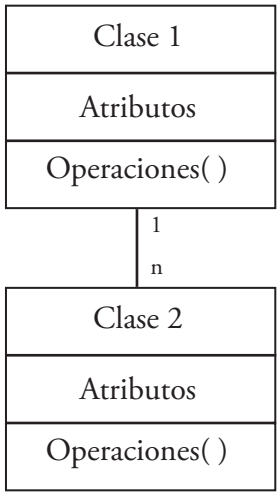
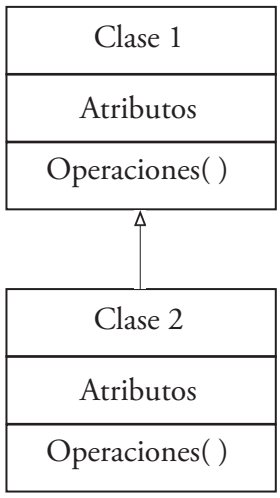
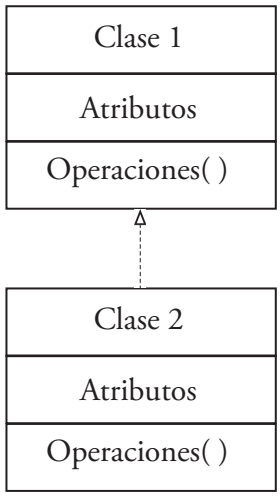
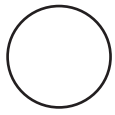
Notación	Descripción
	La *agregación* es una forma especial de asociación que especifica una relación todo-parte entre el agregado (el todo) y un componente (la parte) y (normalmente) control del tiempo de vida. Todas las propiedades de la asociación se aplican a la agregación.
	La *composición* es similar a la agregación pero más completa; el todo conocido como compuesto tiene la responsabilidad explicita de la creación y destrucción de los objetos. La multiplicidad de una composición siempre es 1. El tiempo de vida coincidente entre las partes y el todo.
	La *multiplicidad* se refiere al número de objetos que se permiten que participen en una asociación. Si no se muestra la multiplicidad, el número por default es 1. La multiplicidad puede ser: • Uno o más (1..*) • Cero o más (*) • Número fijo (m)

Tabla C.3. Elementos del diagrama de clases (continuación)

Notación	Descripción
	Una *generalización* es una relación entre un elemento general (llamado superclase o padre) y un caso específico de ese elemento (llamado subclase o hijo). Esta relación representa que una clase define un conjunto de características (especializadas o extendidas) en otra. Implica directamente la herencia de la superclase a la subclase.
	Una *dependencia* es una relación de uso que declara que un cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza. Se definen diecisiete tipos de dependencia, a continuación se describen las más usadas: • Bind: especifica que el origen de la dependencia instancia a la plantilla destino con los parámetros reales dados. • Derive: Especifica que el origen puede calcularse a partir del destino. • Friend: Especifica que el origen tiene una visibilidad especial en el destino.
	Una *interfaz* es un conjunto de operaciones que especifica cierto aspecto de la funcionalidad de una clase, y es el conjunto de operaciones que una clase presenta a otra.
	Los *puertos* son puntos de conexión de las clases y son llamados por los servicios proporcionados y requeridos a través de ellos. Un puerto puede proporcionar o necesitar una interfaz.

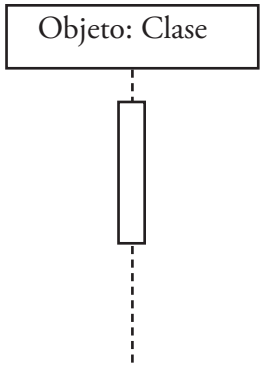
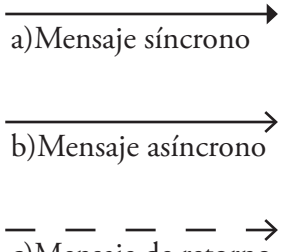
C.1.3. DIAGRAMA DE SECUENCIA

El diagrama de secuencias es un esquema conceptual que permite representar el comportamiento de un sistema, para lo cual emplea la especificación de los objetos que se encuentran en un escenario y la secuencia de mensajes intercambiados entre ellos, con el fin de llevar a cabo una transacción del sistema [Zapata, M. & Garcés, L., 2008].

C.1.3. DIAGRAMA DE SECUENCIA

Los diagramas de secuencia son una representación en el tiempo que resalta la ordenación temporal de los mensajes [Booch, Rumbaugh & Jacobson, 2004] intercambiados entre distintos objetos para lograr un comportamiento particular. Este tipo de diagramas son utilizados para validar los casos de uso, pues estos se usan continuamente como modelos explicativos para los escenarios de los casos de uso [Zapata, M. & Garcés, L., 2008]. Mediante la creación de un diagrama de secuencias con un actor y los elementos involucrados en el caso de uso, puede modelar la secuencia de pasos que toman el usuario y el sistema para completar las tareas requeridas. Para interactuar entre sí, los objetos se envían mensajes. Durante la recepción de un mensaje, los objetos se vuelven activos y ejecutan el método del mismo nombre. Un envío de mensaje es, por tanto, una llamada a un método. Los principales componentes de un diagrama de secuencia se presentan en la Tabla C.4.

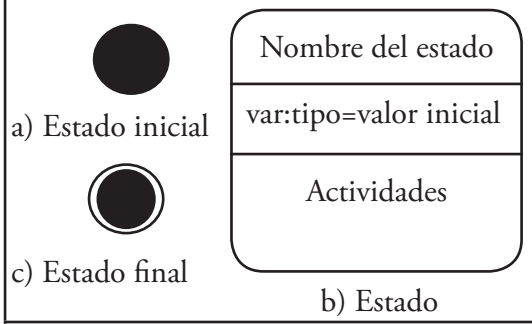
Tabla C.4. Elementos del diagrama de secuencia

Notación	Descripción
	Los *objetos* son instancias de las clases. Un objeto es una estructura de datos que proporciona servicios que actúan sobre datos. Los datos de los objetos se almacenan en forma de atributos, variables (locales simples o primitivas del objeto). Los servicios que actúan sobre los datos son llamados métodos, los cuales son servicios invocados por otros objetos o por métodos existentes en otro objeto. Se representa en un rectángulo y con su texto subrayado. A cada instancia se asocia una línea de vida (línea vertical punteada) que muestra las acciones y reacciones de la misma, así como los periodos durante los cuales ésta está activa (rectángulo), es decir, durante los que ejecuta uno de sus métodos.
	Un *mensaje* es un mecanismo por medio del cual se comunican los objetos. Los envíos de mensajes se representan mediante flechas horizontales que unen la línea de vida del objeto emisor con la línea de vida del objeto destinatario. El objetivo del mensaje es que el objeto emisor haga una llamada al a una operación del objeto receptor. El orden del tiempo se asigna de arriba abajo. Existen tres tipos de mensajes: síncrono y asíncrono. • El mensaje síncrono es cuando el expedidor del mensaje espera que la activación del método mencionado por el destinatario finalice antes de continuar su actividad. • El mensaje asíncrono el expedidor no espera el término de la activación invocada por el destinatario. • El mensaje de retorno indica el regreso de un mensaje.

C.1.4. DIAGRAMA DE ESTADO

Un diagrama de estados muestra una máquina de estados, destacando el flujo de control entre estados [Booch, Rumbaugh & Jacobson, 2004]. Describen todos los estados posibles en los que puede entrar un objeto particular y la manera en que cambia el estado del objeto, como resultado de los eventos que llegan a él. Así este tipo de diagramas representa el conjunto de estados que un objeto puede experimentar y las acciones que llevan un objeto de un estado a otro. Por lo que son especialmente importantes para describir el comportamiento de un sistema cuyo comportamiento está dirigido por eventos. El diagrama de estados tiene como componentes principales: los estados y las transiciones (que incluye a su vez: eventos, acciones y actividades) (véase Tabla C.5).

Tabla C.5. Elementos del diagrama de estado

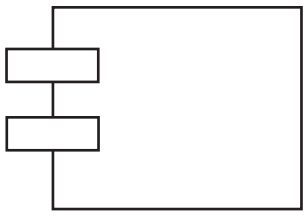
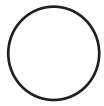
Notación	Descripción
	Un *estado* (Figura b) es la condición de un objeto en un momento determinado y especifica la respuesta de un objeto a los posibles eventos de entrada. Está definido por los valores de los atributos del objeto. Los estados también pueden indicar el comienzo (Figura a) del diagrama de estado, o bien el final del mismo (Figura c).
	Un *evento* es un acontecimiento importante a tomar en cuenta para el sistema. Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser: • Evento cambio: condición que toma el valor de verdadero o falso. • Evento señal: recepción de una señal explicita de un objeto a otro. • Evento llamada: recepción de una llamada a una operación.
	Una *transición* es una relación entre dos estados, indica que, cuando ocurre un evento el objeto pasa del estado anterior al siguiente. Una transición se dispara cuando ocurre un evento.

C.1.5. DIAGRAMA DE COMPONENTES

Los diagramas de componentes representan una colección de componentes físicos y las relaciones de un sistema; representando así la estructura física de la implementación. Este tipo de diagramas se basa en los componentes. Uno de los aspectos más importantes de usar componentes es su reutilización.

Los Diagramas de componentes ilustran las piezas del software (cualquier unidad software, por ejemplo: archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables), entidades grandes y abstractas (una maquina virtual), controladores empotrados, etc. que conformarán un sistema. Un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clase. Un diagrama de componentes contiene, obviamente, componentes, interfaces y relaciones (véase Tabla C.6).

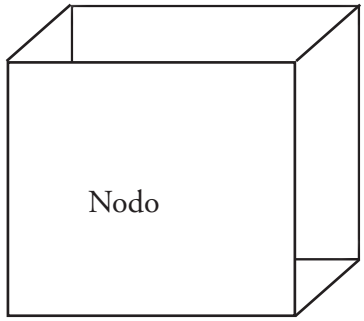
Tabla C.6. Elementos del diagrama de componentes

Notación	Descripción
	Un *componente* es una unidad de software que ofrece una serie de servicios a través de una o varias interfaces. Se trata de una caja negra cuyo contenido queda fuera del interés de los clientes. Un componente posee características similares a una clase: tiene nombre, realiza interfaces, puede participar de relaciones, puede tener instancias, puede participar en interacciones. Solo que un componente representa un elemento físico (bits) y una clase es una abstracción lógica. Las operaciones de un componente solo se alcanzan a través de interfaces. Existen principalmente tres tipos de componentes: • Componentes de distribución, que conforman el fundamento de los sistemas ejecutables (DLL, ejecutables, etc.). • Componentes para trabajar en el producto, a partir de los cuales se han creado los componentes de distribución (archivos de base de datos y de código). • Componentes de ejecución, creados como resultados de un componente de ejecución. Usualmente los componentes son implementados por una o más clases (u objetos) en tiempo de ejecución.
	La *interfaz* contiene un conjunto de operaciones y se utiliza para especificar los servicios de un componente. Una interfaz se conecta al componente que la implementa a través de una relación de realización, y al componente que utiliza sus servicios con una dependencia.

C.1.6. DIAGRAMA DE DESPLIEGUE

Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Por lo tanto representa la visualización de los componentes sobre los dispositivos físicos. Estos diagramas se componen por: nodos, relaciones (dependencia, generalización, asociación, y realización) y puede contener los componentes que residen en los nodos (véase Tabla C.7).

Tabla C.7. Elementos del diagrama de despliegue

Notación	Descripción
	Un *nodo* es un objeto físico en tiempo de ejecución que representa un recurso computacional, que normalmente tiene algo de memoria y, a menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representa típicamente un procesador o un dispositivo sobre el que se pueden desplegar los componentes. Por lo que se considera el elemento primordial del hardware. Los componentes se ejecutan en los nodos.

C.2. CARACTERISTICAS DE UML PARA SE

UML está estructurado por un conjunto de propiedades clave propias para el diseño de sistemas empotrados, entre las que podemos mencionar las siguientes:

- Con los diagramas de clases se realiza el análisis y diseño de los sistemas, creándose el diseño conceptual de la información que se manejará en el sistema, los componentes, y la relación entre uno y otro. Permitiendo que el sistema empotrado a desarrollar pueda ser descrito mediante una estructura estática apoyada en clases, interfaces y colaboraciones, y las relaciones entre ellas.
- Mediante el diagrama de casos de uso se establece lo que se requiere que el sistema haga.
- Apoyados en la vista estructural se puede establecer la manera como el sistema deberá estar estructurado para satisfacer los requisitos.
- Para realizar el análisis del sistema se toma como base a UML, ya que define un marco de trabajo para capturar y expresar modelos de objetos y metamodelos con semántica genérica y notación gráfica. Los metamodelos permiten capturar conceptos tan importantes como el de Tiempo Real, Concurrencia y Seguridad.
- Los Diagramas de Despliegue: Involucran software que controlan dispositivos tales como motores, actuadores, y monitores, y que en su momento, es controlado por un estimulo externo tal como sensores de entrada, movimiento y cambios en la temperatura. Los diagramas de despliegue facilitan la comunicación entre los ingenieros de hardware del proyecto y los desarrolladores de software.
- Los Diagramas de Despliegue Son útiles para razonar acerca de los compromisos entre el hardware y el software. Los diagramas de despliegue se utilizan para visualizar, especificar, construir y documentar las decisiones de ingeniería del sistema. Los diagramas de despliegue tratan de capturar la topología de un sistema hardware. De este modo, la combinación de los diagramas de componentes y despliegue muestran las relaciones físicas entre los componentes de software y de hardware del sistema.

- UML cuenta con los diagramas necesarios para modelar la gestión del tiempo y de recursos que caracterizan a los sistemas empotrados.
- La notación usada por UML es de naturaleza gráfica, fácil de dominar y simple de entender.

APÉNDICE D
BASE DE CONOCIMIENTO DE SPIES

Artefacto ANAREQ – **Informe de análisis sobre los requisitos** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Descripción general Perspectiva del producto	
Funcionalidad del sistema	
1.	
2.	
3.	
4.	
5.	
Características de los usuarios	
Tipo de usuario:	
Formación:	
Habilidades:	
Restricciones	
1.	
2.	
3.	
4.	
5.	
Suposiciones y dependencias	
1.	
2.	
3.	
4.	
5.	
Evolución previsible del sistema	

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**
Fecha: _____

Requisitos específicos				
Número de requisito:				
Nombre del requisito:				
Tipo:	Operacional	Funcional	Diseño	Restricción
Fuente del requisito:				
Prioridad:	Alta/Esencial	Media/Deseado	Baja/Opcional	
Requisitos comunes de las interfaces Interfaces de usuario				
Interfaces de hardware				
Interfaces de software				
Interfaces de comunicación				
Requisitos funcionales				

Artefacto ASIREQ – **Asignación de requisitos por componente de producto (continuación)**
Fecha: _____

Requisitos no funcionales Requisitos de rendimiento		
Seguridad		
Otros requisitos		
Componentes del sistema		
Nombre de componente:		
Descripción:		
Requisitos que se añaden al componente:		
1.		
2.		
3.		
4.		

Tabla D.1. Instrucciones para el artefacto Asignación de requisitos por componente de producto

Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Descripción general Perspectiva del producto	Indicar si es un producto independiente o parte de un sistema mayor. En el caso de tratarse de un producto que forma parte de un sistema mayor, se deberá añadir un diagrama que sitúe el producto dentro del sistema e identifique sus conexiones para facilitar la comprensión.
Funcionalidad del sistema	Resumen de las funcionalidades principales que el sistema debe realizar, sin entrar en información de detalle. En ocasiones la información de esta sección puede tomarse de un documento de especificación del sistema de mayor nivel (Artefacto DREQ). Las funcionalidades deben estar organizadas de manera que el cliente o cualquier interlocutor puedan entenderlo perfectamente. Para ello se pueden utilizar métodos textuales o gráficos. NOTA: Se deberán agregar tantas filas como funcionalidades se requieran.
Características de los usuarios	Descripción de los usuarios del producto, incluyendo nivel educacional, experiencia y experiencia técnica. NOTA: Se deberán agregar tantas tablas como número de usuarios.
Restricciones	Descripción de aquellas limitaciones a tener en cuenta a la hora de diseñar y desarrollar el sistema, tales como el empleo de determinadas metodologías de desarrollo, lenguajes de programación, normas particulares, restricciones de hardware, de sistema operativo etc. NOTA: Se deberán agregar tantas filas como restricciones del sistema
Suposiciones y dependencias	Descripción de aquellos factores que, si cambian, pueden afectar a los requisitos. Por ejemplo una suposición puede ser que determinado sistema operativo está disponible para el hardware requerido. De hecho, si el sistema operativo no estuviera disponible, se deberá registrar en el análisis de los requisitos y éstos deberían modificarse.
Evolución previsible del sistema	Identificación de futuras mejoras al sistema, que podrán analizarse e implementarse en un futuro.
Requisitos específicos	Esta es la sección más extensa y más importante del documento. Debe contener una lista detallada y completa de los requisitos que debe cumplir el sistema a desarrollar. El nivel de detalle de los requisitos debe ser el suficiente para que el equipo de desarrollo pueda diseñar un sistema que satisfaga los requisitos y los encargados de las pruebas puedan determinar si éstos se satisfacen. La fuente de estos requisitos será el Diagrama de Requisitos (Artefacto DREQ). Los requisitos se dispondrán en forma de listas numeradas para su identificación, seguimiento, trazabilidad y validación utilizando la nomenclatura especificada por la actividad específica 1 de la práctica 1.2 del área de proceso Definición de los Requisitos. Para cada requisito debe completarse una tabla.

Tabla D.1. Instrucciones para el artefacto Asignación de requisitos por componente de producto (continuación)

Requisitos comunes de las interfaces Interfaces de usuario	Describir los requisitos del interfaz de usuario para el producto. Esto puede estar en la forma de descripciones del texto o pantallas del interfaz. Por ejemplo posiblemente el cliente ha especificado el estilo y los colores del producto. Describa exactamente cómo el producto aparecerá a su usuario previsto.
Interfaces de hardware	Especificar las características lógicas para cada interfaz entre el producto y los componentes de hardware del sistema. Se incluirán características de configuración.
Interfaces de software	Indicar si es necesario integrar el sistema con otros productos de software. Para cada producto de software debe especificarse lo siguiente: • Descripción del producto software utilizado • Propósito del interfaz • Definición del interfaz: contenido y formato
Interfaces de comunicación	Describir los requisitos del interfaces de comunicación si hay comunicaciones con otros sistemas y cuales son los protocolos de comunicación.
Requisitos funcionales	Definición de acciones fundamentales que debe realizar el software al recibir información, procesarla y producir resultados. En ellas se incluye: • Comprobación de validez de las entradas • Secuencia exacta de operaciones • Respuesta a situaciones anormales (desbordamientos, comunicaciones, recuperación de errores) • Parámetros • Generación de salidas • Relaciones entre entradas y salidas (secuencias de entradas y salidas, formulas para la conversión de información) • Especificación de los requisitos lógicos para la información que será almacenada en base de datos (tipo de información, requerido) NOTA: Se deberá agregar la especificación de cada requisito funcional.
Requisitos no funcionales Requisitos de rendimiento	Especificación de los requisitos relacionados con la carga que se espera tenga que soportar el sistema. Por ejemplo, el número de terminales, el número esperado de usuarios simultáneamente conectados, número de transacciones por segundo que deberá soportar el sistema, etc. Todos estos requisitos deben ser mesurables. Por ejemplo, indicando “el 95% de las transacciones deben realizarse en menos de 1 segundo”, en lugar de “los operadores no deben esperar a que se complete la transacción”.

Tabla D.1. Instrucciones para el artefacto Asignación de requisitos por componente de producto (continuación)

Seguridad	Especificación de elementos que protegerán al software de accesos, usos y sabotajes maliciosos, así como de modificaciones o destrucciones maliciosas o accidentales. Los requisitos pueden especificar: • Empleo de técnicas criptográficas. • Registro de ficheros con "logs" de actividad. • Asignación de determinadas funcionalidades a determinados módulos. • Restricciones de comunicación entre determinados módulos. • Comprobaciones de integridad de información crítica.
Mantenibilidad	Identificación del tipo de mantenimiento necesario del sistema. Especificación de quién debe realizar las tareas de mantenimiento, por ejemplo usuarios, o un desarrollador. Especificación de cuándo deben realizarse las tareas de mantenimiento. Por ejemplo, generación de estadísticas de acceso semanal y mensual.
Otros requisitos	Cualquier otro requisito que no encaje en ninguna de las secciones anteriores. Por ejemplo: • Requisitos culturales y políticos. • Requisitos Legales.
Componentes del sistema	Junto con la descripción del Diagrama de Requisitos () identificar componentes tentativos del sistema. Esta arquitectura será refinada más adelante por el área de proceso de Diseño del Producto.

Artefacto DACT – **Diagrama de actividades** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Artefacto DCUF – **Diagrama de casos de uso funcional** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Artefacto DESC – **Diagrama de secuencia** **Fecha: _____**

Nombre del proyecto:	
Nombre del responsable:	

Artefacto DEST – **Diagrama de estructura** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Artefacto DMAQ - **Diagrama de maquinas de estado** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Artefacto DREQ – **Diagrama de requisitos** **Fecha: _____**

Nombre del proyecto:	
Nombre del responsable:	

Artefacto ESF – **Esfuerzo del Proyecto** **Fecha: _____**

Nombre del proyecto:	
Nombre del responsable:	

Tamaño del producto		**Planificado**	**Actual**
Diseño Alto Nivel			
Diseño Bajo Nivel			
Páginas Requisitos			
Páginas de texto			
LOC Base (medido)			
LOC Borrado			
LOC Modificado			
LOC Añadido			
LOC Reutilizado			
LOC Total / Nuevo-cambiado			
LOC Total			
LOC Total / Nuevo-reutilizado			

Tiempo por Fases	**Planificado**	**Actual**	**% Actual**
Planificación			
Requisitos			
Plan de Pruebas del Sistema			
Diseño de Alto Nivel			
Plan de Pruebas de Integración			
Inspección Diseño de Alto Nivel			
Planificación de la Implementación			
Diseño detallado			
Revisión del Diseño detallado			
Desarrollo de Pruebas			
Inspección Diseño Detallado			
Código			
Revisión de Código			
Compilación			
Inspección de Código			
Prueba Unitarias			
Pruebas de Construcción e Integración			
Prueba de Sistema			
Documentación			
Totales			

Artefacto ESF – **Esfuerzo del Proyecto (continuación)** **Fecha:**_____ **Ciclo:**_____

Nombre del proyecto:	
Nombre del responsable:	

Defectos Inyectados		**Planificado**	**Actual**
Requisitos			
Plan de Pruebas del Sistema			
Diseño de Alto Nivel			
Plan de Pruebas de Integración			
Inspección Diseño de Alto Nivel			
Planificación de la Implementación			
Diseño detallado			
Revisión del Diseño detallado			
Desarrollo de Pruebas			
Inspección Diseño Detallado			
Código			
Revisión de Código			
Compilación			
Inspección de Código			
Prueba Unitarias			
Pruebas de Construcción e Integración			
Prueba de Sistema			
Desarrollo Total			
Defectos Removidos	**Planificado**	**Actual**	**% Actual**
Requisitos			
Plan de Pruebas del Sistema			
Diseño de Alto Nivel			
Plan de Pruebas de Integración			
Inspección Diseño de Alto Nivel			
Diseño detallado			
Revisión del Diseño detallado			
Desarrollo de Pruebas			
Inspección Diseño Detallado			
Código			

Artefacto ESF – **Esfuerzo del Proyecto (continuación)** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Defectos Removidos	**Planificado**	**Actual**	**% Actual**
Inspección de Código			
Prueba Unitarias			
Pruebas de Construcción e Integración			
Prueba de Sistema			
Desarrollo Total			

Tabla D.2. Instrucciones para el artefacto Esfuerzo del Proyecto

Objetivo	Este artefacto es útil para determinar los atributos del producto, fijar y determinar el esfuerzo y tiempo destinado para al proyecto.
General	El artefacto ESF es útil para
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Tamaño del producto	Indica el tamaño del producto en base a diferentes criterios, entre ellos: diseño de alto nivel, diseño de bajo nivel, líneas de código (LOC), páginas de requisitos, etc. Este apartado tiene dos columnas: Planificado (para registrar el progreso que se espera del desarrollo de software) y el Actual (para registrar el progreso que verdadero del desarrollo de software). Se compone por los siguientess campos: • LOC Base (medido): indica las líneas de código necesarias-suficientes para la realización del proyecto. • LOC Borrado: para registrar el número de líneas de código que han sido borradas durante el desarrollo del sistema. • LOC Modificado: el número de líneas de código que han sido modificadas durante el desarrollo del sistema. • LOC Añadido: para indicar la cantidad de código que fue incrementado al sistema. • LOC Reutilizado: SPIES se basa en el desarrollo basado en componentes, lo que implica la reutilización de código. Util para estimar la tasa de reutilización de código previamente desarrollado.
Tiempo por Fases	Este apartado sirve para registrar el tiempo empleado para realizar cada uno de las actividades propuestas por SPIES. Se compone por tres columnas: planificado (para registrar el tiempo en horas en que se ha pensado llevar a cabo la tarea), actual (para registrar el tiempo gastado en horas para llevar a cabo la tarea) y % actual (). Esto permite realizar mejores estimaciones de tiempo; se obtiene información para proyectar la distribución de tiempo de un nuevo proyecto, basado en la distribución de tiempo de proyectos anteriores.
Defectos Inyectados	Permite realizar estimaciones de calidad, pues proporciona información para proyectar la distribución de defectos de un nuevo proyecto, basado en la distribución de defectos de proyectos anteriores. Permite también identificar las fases problemáticas y trabajar para mejorarlas.
Defectos Removidos	Permite realizar estimaciones de calidad.

Artefacto HAB – **Planificación del conocimiento y de las habilidades** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Evaluación del trabajo requerido					
	Estimado			Real	
	Trabajo requerido (%)			Trabajo requerido (%)	
Equipo de diseño					
Equipo de hardware					
Equipo de integración					
Equipo de software					
Equipo mecánico					
Evaluación del conocimiento para ejecutar el proyecto					
Equipo de diseño	1	2	3	4	5
Equipo de hardware	1	2	3	4	5
Equipo de integración	1	2	3	4	5
Equipo de software	1	2	3	4	5
Equipo mecánico	1	2	3	4	5
Otro:	1	2	3	4	5
Inventario de conocimiento/habilidades disponibles					
Conocimiento/Habilidades	Disponible en el equipo			Responsable	
Inventario de conocimiento/habilidades no disponibles					
Conocimiento/Habilidades	Área			Responsable	
Evaluación del equipo					
Conocimiento que debe adquirirse	1	2	3	4	5
Productividad del equipo requerida	1	2	3	4	5
Calidad del proceso	1	2	3	4	5
Calidad del producto	1	2	3	4	5
Equipo mecánico	1	2	3	4	5
Otro:	1	2	3	4	5
Mecanismos para proporcionar el conocimiento y habilidades necesarias					
	Software	Hardware	Diseño	Integración	Mecánico
Formación interna					
Formación externa					
Nuevas adquisiciones					
Adquisición externa de habilidades					

Tabla D.3. Instrucciones para el artefacto Estrategia de desarrollo

Objetivo	Este artefacto tiene como objetivo principal determinar las habilidades y conocimientos necesarios para ejecutar el proyecto.
General	Es útil para identificar las habilidades y conocimientos requeridas no disponibles en el equipo y que son vital para el desarrollo del SE.
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Evaluación del trabajo requerido	Esta área tiene el propósito de analizar el desarrollo del SE y determinar qué porcentaje será realizado por los cada una de las áreas. Cuenta con dos columnas: • Trabajo requerido estimado: se debe introducir el porcentaje del trabajo que se estima realizar por el equipo para desarrollar el SE. • Trabajo requerido real: se debe introducir el porcentaje del trabajo actual que se ha realizado por el equipo para desarrollar el SE.
Evaluación del conocimiento para ejecutar el proyecto	Esta apartado tiene el propósito de obtener una estimación del nivel de conocimiento necesario para desarrollar el SE. Cada uno de los equipos debe encerrar en un círculo el nivel de conocimiento con el que deben contar para realizar sus tareas. Con el número 5 se indica que el nivel de conocimiento del equipo debe ser “alto”.
Inventario de conocimiento/ habilidades disponibles	Esta área tiene el propósito de realizar un inventario de los conocimiento y habilidades que se requieren para realizar el SE. • Conocimiento/Habilidades: en esta columna se debe introducir el conocimiento o habilidad necesaria para realizar el proyecto. • Disponible en el equipo: para indicar el equipo que cubrirá el conocimiento o habilidad. • Responsable: se debe llenar con el nombre del encargado de administrar el conocimiento o habilidad.
Inventario de conocimiento/ habilidades no disponibles	Esta área tiene el propósito es saber conque conocimientos y habilidades no se cuenta para realizar el SE. • Conocimiento/Habilidades: en esta columna se debe introducir el conocimiento o habilidad no disponible para realizar el proyecto. • Área: en esta columna se debe indicar el equipo que carece del conocimiento o habilidad. • Responsable: se debe llenar con el nombre del encargado de administrar el conocimiento o habilidad.
Evaluación del equipo	Esta apartado tiene el propósito de evaluar el equipo para desarrollar el SE. El equipo debe calificarse en cada una de las evaluaciones. Con el número 5 se indica que el nivel de conocimiento del equipo debe ser “alto”.

Artefacto PRY - **Plan del proyecto** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Tarea		Horas planificadas	Horas reales		
Fase	Nombre de la tarea	Total horas	Horas estimadas	Horas acumuladas	Valor real
1	**Planificación**				
	Establecer estimaciones				
	Estimar alcance del proyecto				
	Establecer estimaciones para productos y tareas				
	Determinar estimaciones de costo y esfuerzo				
	Desarrollar el plan del proyecto				
	Establecer presupuesto y calendario				
	Identificar los riesgos del proyecto				
	Planificar la participación de los proveedores de requisitos				
	Establecer el plan del proyecto				
2	**Especificación de requisitos**				
	Desarrollar los requisitos del producto				
	Obtener las necesidades				
	Desarrollar los requisitos				
	Establecer los requisitos del producto y de componentes del producto				
	Identificar los requisitos de la interfaz				
	Analizar y validar los requisitos				
	Establecer los conceptos operativos y los escenarios				
	Establecer una definición de funcionalidad requerida				
	Analizar y validar los requisitos				
3	**Diseño del producto**				
	Diseñar la arquitectura del sistema				
	Diseñar los diagramas de estructura				
	Asignar las funciones entre los subsistemas				
	Crear la arquitectura de los subsistemas				
	Diseñar el diagrama del modelo de objetos				
	Generar el código y compilar el modelo				
	Crear los diagramas de secuencia				

Tabla D.4. Instrucciones para el artefacto Plan del proyecto

Objetivo	Este artefacto es usado para analizar las actividades que forma el plan de proyecto, registrar el total de horas planeadas e indicar el valor en horas utilizadas para realizar cada una de las actividades.
General	El artefacto debe ser llenado por el grupo que desarrollará el Sistema Empotrado
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto PRY; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Fase	La columna "Fase" es el número que corresponde con las fases que integran la metodología SPIES. Las fases de SPIES son ocho: (1) Planeación, (2) Especificación de requisitos, (3) Diseño del producto, (4) Desarrollo del producto, (5) Integración del producto, (6) Validación del producto, (7) Entrega y mantenimiento y (8) Mejora continua.
Tarea	Contiene la lista de tareas que deben realizarse de acuerdo a cada una de las fases que integran a SPIES.
Horas planificadas	Es el tiempo en horas calculado para cumplir con una tarea. • Total horas: en esta columna se debe indicar el total de horas destinadas para cumplir con la tarea en curso.
Horas reales	Son las horas actuales y/o efectivas que se deben asignar a cada tarea. • Horas estimadas: En esta columna se debe introducir el valor que se considera propio para realizar la tarea. • Horas acumuladas: debe ser llenada conforme se realizan cada una de las actividades, corresponde al total de tiempo en horas gastadas en realizar la actividad. • Valor real: En esta columna se introduce el valor efectivo en horas empleado en realizar la actividad. Se debe obtener usando la columna de "Horas acumuladas"

Artefacto RIE – **Riesgos del proyecto** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Fecha:

Riesgo/ Problema	Número	Prioridad	Propietario	Fecha RE	Resuelto

Descripción:

Fecha:

Riesgo/ Problema	Número	Prioridad	Propietario	Fecha RE	Resuelto

Descripción:

Fecha:

Riesgo/ Problema	Número	Prioridad	Propietario	Fecha RE	Resuelto

Descripción:

Fecha:

Riesgo/ Problema	Número	Prioridad	Propietario	Fecha RE	Resuelto

Descripción:

Tabla D.5. Instrucciones para el artefacto Riesgos del proyecto

Objetivo	Este artefacto es una guía para identificar y analizar sobre posibles riesgos que pueden surgir en el desarrollo del SE.
General	El artefacto RIE es útil para registrar la información de posibles riesgo que proceden de revisiones, compilaciones, integración, pruebas, etc.
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Fecha	Campo para indicar la fecha en la que se encontro y corrigio el riesgo/ problema.
Riesgo/ Problema	En esta columna se escribe el riego o problema identificado.
Número	Es un número único para identificar el riesgo/problema.
Prioridad	Útil para indicar la prioridad del riesgo/problema a partir de un estándar establecido por el equipo de desarrollo.
Propietario	Para registrar la procedencia del riesgo o problema.
Fecha RE	Útil para anotar la fecha en que el riesgo o problema fue resuelto.
Resuelto	Se debe de indicar si el riesgo o problema ha sido resuelto.
Descripción	En este apartado se detalla el riego o problema.

Artefacto SCHE – **Calendario del Proyecto** **Fecha:** _________

Nombre del proyecto:	
Nombre del responsable:	

		Planificado			Actual		
No. Semana	Fecha	Horas planificadas	Horas acumuladas	Valor planeado acumulativo	Horas equipo	Horas acumuladas	Valor ganado por semana

Tabla D.6. Instrucciones para el artefacto Calendario del Proyecto

Objetivo	El propósito de este artefacto es servir de guía para desarrollar un calendario basado en estimaciones de proyectos anteriores.
General	El artefacto SCHE sirve para registrar el tiempo trabajado en el desarrollo de un SE.
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
No. Semana	Se debe escribir el número de semana para que se calcularan o se registren los datos
Fecha	Es este apartado se debe registrar la fecha de la semana a la que se hace referencia
Planificado	Horas planificadas: En esta columna se debe introducir el valor que se considera propio para realizar la tarea. Horas acumuladas: es el total de horas planificadas que se gastarán al realizar la actividad. Valor planeado acumulativo: Es el total de tiempo planeado que se acumulará.
Actual	Horas equipo: En esta columna se introduce el valor efectivo en horas empleado por el equipo para realizar las tareas, en la semana indicada. Horas acumuladas: debe ser llenada conforme se realizan cada una de las actividades, corresponde al total de tiempo en horas gastadas en realizar la actividad. Valor ganado por semana: Es el tiempo que no se gasto en realizar las actividades en la semana.

Artefacto STR – **Estrategia de desarrollo** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Descripción general de la funcionalidad del sistema

Componentes o módulos del sistema		
	Componente	Descripción funcional
1.		
2.		
3.		
4.		

Componentes a reutilizar (solo si existen proyectos similares anteriores)		
Nombre del Proyecto:		
Responsable:		
	Componente	Descripción funcional
1.		
2.		
3.		
4.		

Componentes a adquirir de forma externa (solo si existe el desarrollo conjunto)		
Responsable:		
	Componente	Descripción funcional
1.		
2.		
3.		
4.		

Posibles riesgos del proyecto

Tabla D.7. Instrucciones para el artefacto Estrategia de desarrollo

Objetivo	Este artefacto define una guía para el desarrollo del producto al obtener la funcionalidad del SE e identificar y dividir el sistema en módulos o componentes.
General	Este artefacto trabaja directamente con los requisitos del proyecto para sugerir una forma de describir al sistema en función de sus componentes. Es importante que participen en su elaboración los desarrolladores de hardware y de software.
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Descripción general de la funcionalidad del sistema	En esta área se debe describir el conjunto de características del SE que lo harán practico y utilitario.
Componentes o módulos del sistema	Un componente es una unidad que posee un conjunto de requisitos que debe ser desarrollado e incorporado al sistema. Algunos ejemplos de componentes son: componentes actuadores, componentes de comunicación componentes de control, componentes interface, componentes sensores, etc.
Componente	En esta columna se deben listar los nombres de los componentes del SE ha desarrollar. Los nombres de los componentes deben describir la tarea que desempeñaran en el sistema.
Descripción funcional	Registra el conjunto de características que harán que el componente sea practico y utilitario para el SE. Se considera como el resultado de desarrollar el componente.
Componentes a reutilizar	Consiste en utilizar a componentes ya existentes, con la función que desempeñaba anteriormente o con otros fines. Tiene el objetivo de maximizar el uso recurrente de componentes ya desarrollados para implementar y/o actualizar SE usando activos existentes. • Nombre del Proyecto: Introducir el nombre del proyecto en el cual fueron desarrollados los componentes. • Responsable: Introducir el nombre de la persona que desarrollo el componente
Componentes a adquirir de forma externa	En esta columna se deben registrar los datos de los componentes externos que formaran parte del Sistema Empotrado con el fin de que los integrantes del equipo los identifiquen y conozcan su utilidad. La importancia del nombre del responsable radica en el hecho de saber con qué persona dirigirse si se presenta algún inconveniente.
Posibles riesgos del proyecto	Se deben anotar los problemas potenciales que, de materializarse, podría afectar el éxito del proyecto.

Artefacto TAM – **Tamaño del Producto** **Fecha:** _____

Nombre del proyecto:	
Nombre del responsable:	

Nombre del Producto	Base	Eliminado	Modificado	Reutilizado	Total	Total Nuevo/ Reutilizado
Totales						

Tabla D.8. Instrucciones para el artefacto Estrategia de desarrollo

Objetivo	Este artefacto es útil para determinar los atributos del producto y fijar el esfuerzo y tiempo destinado para realizarlos.
General	El artefacto TAM es útil para realizar un seguimiento del tamaño de los productos generados por el desarrollo de un SE. El objetivo es calcular el tamaño e indirectamente el esfuerzo a emplearse en el proyecto.
Encabezado	Se debe indicar el nombre del integrante del equipo responsable de registrar los datos en el artefacto; el nombre del proyecto, la fecha de llenado del artefacto (usando el formato de día/mes/año) y la versión del documento para controlar modificaciones.
Nombre del producto	En esta columna se debe introducir el nombre del producto (por ejemplo: artefacto PRY).
Base	Se debe indicar un valor que represente el tamaño del producto sobre el que se va a trabajar. Se aconseja indicar el tamaño de acuerdo a la cantidad de hojas del producto.
Eliminado	En esta columna se debe indicar la cantidad del tamaño prescindido del producto propuesto en la columna “Base”.
Modificado	Se debe indicar la cantidad del tamaño modificado o transformado del producto Base.
Reutilizado	Esta columna es para indicar la cantidad de producto reutilizado para realizar el producto.
Total	Esta columna depende de los datos introducidos en las columnas anteriores, su valor se obtiene al considerar el valor del producto base, eliminado, modificado y reutilizado. Total = *Base+Reutilizado-Eliminado*
Total Nuevo/Reutilizado	Esta columna se debe anotar la cantidad de producto nuevo y la cantidad de producto reutilizado.
Totales	Es el resultado de la suma de la columna.