UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
POSTGRADO EN COMPUTACIÓN

"ESPECIFICACIÓN DE FRAGMENTOS SITUACIONALES DE UN MÉTODO ÁGIL Y DISCIPLINADO[1] PARA LA IMPLEMENTACIÓN DE SOFTWARE"

Autor: Ing. Arístides Castillo C.

Tutor: Dra. Judith Barrios A.

Trabajo de grado presentado ante la Ilustre Universidad de Los Andes como requisito parcial para optar al título de: *Magíster Scientiae en Computación*

Mérida, Octubre 2010

[1] A los efectos de este trabajo de grado, disciplina se entenderá como la conjunción de las acepciones de conformidad con los procesos y estándares establecidos y auto-control

Dedicatoria:

A mi esposa Ana Rosa quien me motiva día a día a reinventarme y convertirme en un mejor hombre, y que pacientemente escuchó los distintos argumentos que usé en este trabajo.

AGRADECIMIENTOS

A Dios, por darme la oportunidad de cumplir mí meta de realizar mi maestría en la ciudad de Mérida.

A mis padres Irene y Máximo, y mis hermanos Ramax, Maxir y Cindha, quienes desde el primer momento en que les propuse mi travesía para realizar mis estudios de postgrado, me apoyaron incondicionalmente.

A mi tutora, Dra. Judith Barrios A., por los excelentes momentos de discusión y reflexión, por ser tan abierta en compartir sus conocimientos y por creer en mí.

Al proyecto Methodius - Investigación FONACIT Nro. 2005000165, que financió este trabajo de investigación, y que espero saque todo el provecho posible de los resultados de esta investigación.

A la Red DBAccess, donde he aprendido muchísimo, cuyos espacios de reflexión continua entre los apuros del día a día permitieron la articulación de muchos de los argumentos expuestos en este trabajo. Y principalmente, por permitirme experimentar todas las prácticas que en el área de implementación de software he propuesto en este trabajo.

A mis primos, Roberto y Diana, quienes me acogieron en su casa como un hijo más y me permitieron iniciar una agradable estadía en la ciudad de Mérida mientras crecía en lo profesional, en lo académico y en lo personal.

Muy especialmente, a Luisita y Taniana, quienes me han apoyado en múltiples ocasiones y sin quienes cursar estudios en el Postgrado de Computación no sería igual. Se les quiere y aprecia

Y a todos aquellos quienes compartieron un espacio de reflexión y filosofía en temas de mejores prácticas de desarrollo de software, que a veces sin saber y otras más con conocimiento de causa, me ayudaban a entretejer las ideas expuestas en este trabajo.

RESUMEN

Desde hace algunos años, los métodos ágiles y los métodos tradicionales han sido expuestos como adversarios irreconciliables e incompatibles, aun cuando ambos han demostrado ser efectivos en ciertos y determinados proyectos y cuando cada enfoque puede aportar prácticas útiles y comprobadas para la solución de algunos de los muchos problemas que enfrenta la ingeniería de software en su día a día. Por esto, se han realizado diversos esfuerzos para lograr integrar las mejores prácticas de cada enfoque, y así se ha logrado demostrar la compatibilidad de ellos.

Un importante punto de mejora en las propuestas de integración de estos enfoque es la posibilidad de utilizar las prácticas alternativas de ambos enfoques de acuerdo a la situación; esto es, a diferencia de colocar un gran condicional respecto a usar un método ágil o un método disciplinado, poder seleccionar de acuerdo a las circunstancias o situación de cada proyecto una práctica distinta, escogiendo entre múltiples opciones de prácticas ágiles y disciplinadas.

La aplicación de los conceptos propios de la Ingeniería de Métodos Situacionales, permite la creación de fragmentos de métodos que pueden ser ensamblados para la creación de métodos adaptados a las circunstancias de cada proyecto para así aplicar las prácticas más recomendadas para cada situación.

Este trabajo de investigación se propone la creación de los fragmentos situacionales de método que combinen prácticas ágiles y disciplinadas para su aplicación en diferentes situaciones y circunstancias, limitando su alcance al contexto del Proceso de Implementación de Software, que involucra las actividades de diseño detallado, aprovisionamiento e integración de unidades de software denominadas componentes. De igual manera se involucra en este Proceso de Implementación las actividades de verificación y validación (V&V) a nivel de los componentes de software.

Palabras Claves: Proceso de Implementación, métodos ágiles, métodos disciplinados, Ingeniería de Métodos Situacionales, metodología

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

Introducción

Desde hace algunos años, dos enfoques que agrupan diversos métodos de desarrollo de software han estado enfrascados en una confrontación, considerados entre sí oponentes irreconciliables en un juego de suma cero. Por un lado, los novedosos métodos ágiles (Ambler 2002, Ambler 2005, Beck, 2000, Fowler y Highsmith, 2001, Schwaber, 2004) que abogan por la adaptabilidad y capacidad de responder oportunamente al cambio y la búsqueda continua de retroalimentación como mecanismo primario de mitigación de riesgos, en sustitución de documentación extensiva y la definición de procesos complejos y fuertemente elaborados; y por el otro lado, los métodos tradicionales y conservadores que reflejan la necesidad de una fuerte disciplina de procesos y prácticas rigurosas (*IEEE* Std Board, 1990, *IEEE* Std Board, 1998, ISO, 2008, Montilva et al, 2008), que promueven la implementación de procesos estándares y bien definidos y que persiguen la meta de lograr que el desarrollo de software sea predecible y repetible, tal como lo han logrado otros campos de la ingeniería, como la mecánica o la electrónica.

Diversos autores (Weber, 2002, Marchesi, 2002, Hefner, 2005, Gibson, 2006) han documentado casos exitosos de aplicación de métodos enmarcados en uno u otro enfoque donde demuestran haber logrado el propósito de facilitar la producción de software de calidad de una forma costeable. A pesar de esto y del reconocimiento extendido de la importancia de la adaptación de los métodos y procesos de desarrollo de software a la circunstancia de cada proyecto, se sigue observando la posición de aclamar la universalidad de uno u otro método o enfoque (Boehm y Turner, 2003). En (Sheard, 2003) se describe esta situación en el contexto del ciclo vicioso de la vida de las Balas de Plata, mencionando que entre los ejemplos recientes de estas balas de plata preferidas se encuentran justamente el Modelo Integrado de Capacidad y Madurez (*Capacity Maturity Model Integration*, *CMMI*® por sus siglas en inglés) y los métodos ágiles.

Mientras transcurre el enfrentamiento entre estos enfoques, a los que se atribuye su cuota de éxito en sus nichos de proyectos característicos (Boehm y Turner, 2003), algunos autores (Boehm y Turner, 2003, Anderson, 2005, Davis et al, 2005, Dutton y McCabe, 2005, Glazer et al, 2008, McMahon, 2004, Pikkarainen, 2006, Hurtado y Bastarrica, 2006) abogan por el aprovechamiento de las ventajas y prácticas de ambos enfoques en conjunto para así usar la herramienta más adecuada al problema en mano. En particular, en un reporte técnico avalado por el Instituto de Ingeniería de Software (*Software Engineering Institute*, SEI por sus siglas en inglés) (Glazer et al, 2008) se reconoce que efectivamente *CMMI*® y los métodos ágiles pueden ser exitosamente usados en conjunto y se declara el interés por parte del instituto en las experiencias de desarrollo en la que estos enfoques interactúen entre sí.

Un aspecto muy importante a ser resaltado en este punto está asociado al reconocimiento explícito de que las características particulares de un proyecto vienen a incidir en la elección adecuada de prácticas de uno u otro enfoque, por lo que aun cuando en dos proyectos distintos se promueva el uso de prácticas mixtas de los enfoques mencionados, es muy probable que la elección de prácticas varíe entre uno y otro, especialmente si los proyectos tienen características esenciales muy diferentes (Ej.: desarrollo de una aplicación web vs desarrollo de un sistema empotrado).

Aun cuando se cuentan con diversas iniciativas de integración de prácticas de desarrollo ágil con prácticas tradicionales estándar de desarrollo de software y modelos tradicionales de mejora de procesos como *CMMI* (Boehm y Turner, 2003, Anderson, 2005, Hurtado y Bastarrica, 2006, Davis et al, 2005), los métodos por ellos propuestos adolecen de uno de los dos siguientes problemas: 1) Presentan una selección rígida de las prácticas de cada enfoque a ser aplicadas o 2) Indican explícitamente que las prácticas a aplicar pueden depender de las características del proyecto, sin brindar orientación acerca de cuando se sugiere o no el uso de una u otra práctica. La elección de las prácticas alternativas entre los enfoques ágil y tradicional que mejor se adecuen a

un proyecto dado es sin duda un problema complejo, por cuanto su solución no puede ser determinada al momento de diseñar un método, sino al momento de aplicarlo.

Una propuesta interesante de solución al problema antes descrito es la de la Ingeniería de Métodos Situacionales (Brinkkemper, 1996, Ralyte, 2001, Rolland, 2005) que haciendo uso del modelado de procesos orientado a decisiones (Pohl et al, 1994) y de componentes de método llamados *fragmentos,* permite separar el objetivo (intención) de una actividad de la manera (estrategia) de llevarla a cabo, permitiendo así la integración de prácticas específicas que provengan de distintos enfoques para adaptarlos a la medida del proyecto en cuestión. La decisión por una u otra estrategia se toma en base a conocimiento explícito en el modelo orientado a decisiones, porque dicha decisión es parte esencial de este tipo de modelado. Así, quien usa el método tiene un contexto de la razón por la que se toma una decisión y la intención de cada actividad en específico.

El presente trabajo explora el uso de la propuesta de la Ingeniería de Métodos Situacionales para integrar las prácticas de los métodos ágiles y de los métodos tradicionales de desarrollo de software, con el fin de generar una especificación situacional de procesos que guíe al usuario en el desarrollo y adaptación de las prácticas acordes a la implementación de los componentes de software.

El alcance de esta investigación se acota al Proceso de Implementación de Software, el cual se centra en el diseño detallado, aprovisionamiento, codificación, verificación y validación e integración, todo a nivel de componentes de software. Se selecciona el Proceso de Implementación por varias razones, entre las cuales cabe destacar el particular foco que los métodos ágiles dan a este proceso como central en todo el proceso de desarrollo, mientras que los métodos tradicionales y orientados al plan, como se verá más adelante, ponen un énfasis menor en el proceso de implementación. Es por esto que se considera que es justamente allí donde los métodos ágiles pueden aportar más prácticas a los métodos orientados al plan. Otra de las razones está asociada a la experiencia que en dicho proceso tiene el autor a lo largo de su experiencia profesional tanto como programador como liderando equipos de programadores, por lo que esta experticia se ajusta a las posibilidades de los métodos situacionales de ser un vehículo apropiado para la transmisión de conocimiento.

Este trabajo de investigación se divide en ocho (8) capítulos. En el primero se hace una referencia detallada del problema que se propone resolver, así como un esbozo detallado de la situación que se propone. En el segundo capítulo expone el estado del arte en Procesos de Implementación entre los enfoques ágil y orientado al plan, comparándolos y proponiendo un modelo conceptual común que permita la integración de las prácticas recomendadas por ambos enfoques. En el tercer capítulo se presenta el marco teórico asociado a la Ingeniería de Métodos Situacionales, las propuestas de esta disciplina y los lineamientos a utilizar en la estructuración de los fragmentos tanto de productos como de proceso. En el capítulo cuarto se presenta el modelo de productos y en el capítulo quinto se presenta el modelo de procesos ambos en consonancia con el modelo conceptual común a los enfoques ágil y disciplinado que se desarrolló en el Segundo capítulo. Para el sexto capítulo se presenta un caso de implementación de modelos de procesos y productos previamente expuestos, mostrando la forma de utilizar tales modelos. Finalmente el capítulo ocho presenta las conclusiones y recomendaciones de este trabajo.

CAPITULO I: El Problema

El presente capítulo tiene como objetivo hacer la exposición del problema objeto de estudio de esta investigación, establecer los objetivos generales y específicos de la misma, así como presentar la justificación, la metodología usada y los resultados esperado del presente trabajo de investigación.

1 Planteamiento del Problema

Diversos autores han documentado recientemente casos de éxito asociados al enfoque ágil (Weber, 2002, Marchesi, 2002) y al enfoque tradicional orientado al plan (Hefner, 2005, Gibson, 2006) de desarrollo de software, demostrando la pertinencia y la capacidad de cada enfoque para lograr el propósito de facilitar la producción de software de calidad de manera costeable. A pesar de esto, existe un antagonismo marcado entre los proponentes de ambos enfoques que hace que se desdeñe de los casos de éxito del enfoque contrario e insistan en la hegemonía de aquel enfoque que defienden (Boehm y Turner, 2003).

Afortunadamente, existe un conjunto de autores que están promoviendo la integración de ambos enfoques (Boehm y Turner, 2003, Dutton y McCabe, 2005, Glazer et al, 2008, McMahon, 2004, Pikkarainen, 2006), reconociendo el valor que tienen tanto las prácticas de los métodos ágiles como de los métodos orientados al plan, especialmente cuando son aplicadas en aquellos escenarios donde tales prácticas son recomendadas o requeridas. Se ha documentado casos de éxito en (Anderson, 2005, Davis et al, 2005, Hurtado y Bastarrica, 2006) logrando la articulación de métodos que hacen honor a los valores y principios del Manifiesto Ágil (Agile Spain, 2005) a la vez que cumplen con los preceptos del Modelo Integrado de Capacidad y Madurez (*CMMI*®).

Entre los autores que promueven la integración de los enfoques ágil y orientado al plan, en especial (Boehm y Turner, 2003), se menciona que el espectro de proyectos donde se pueden usar las prácticas ágiles o tradicionales difícilmente se trata de una clasificación entre blanco y negro, sino que existe toda una escala intermedia de grises donde no necesariamente una práctica asociada a alguno de los enfoques resulta ser siempre la mejor elección en todos los casos.

Se observa en las referencias que proponen métodos que integran prácticas de los enfoques ágil y tradicional anteriormente citadas, que solamente se hace una integración estática de una selección específica de prácticas. Esto tiene la desventaja de no tomar en cuenta aquellas prácticas que persiguen el mismo fin en el enfoque complementario con las que bien vale la pena contar entre las herramientas disponibles a la hora de encarar un proyecto de desarrollo de software, que pudiesen adaptarse a un contexto específico de proyecto donde las seleccionadas pudiesen no hacerlo. No hay motivos para pensar que tal selección estática de prácticas integradas de ambos enfoques si puede adaptarse a cada distinta situación de un proyecto de desarrollo dado o que una práctica particular que no formó parte de la selección inicial no pueda ser útil a algún proyecto de desarrollo de software distinto. Es importante contar con mecanismos que permitan la selección entre varias prácticas que persigan el mismo objetivo, usando como discriminante la situación o circunstancia del problema que se quiere resolver.

Estas deficiencias en la creación de métodos adaptables a la situación particular de un proyecto buscan ser solventadas por la Ingeniería de Métodos Situacionales (Brinkkemper, 1996, Ralyte, 2001, Rolland, 2005), en particular haciendo uso del Modelado de Procesos Orientado a Decisiones (Pohl et al, 1994), donde las decisiones asociadas a la selección de la manera (estrategia) en que un objetivo se debe lograr (intención) son explícitas en el modelo mismo del proceso, favoreciendo una mayor y mejor difusión y transmisión del conocimiento desde quien diseña el proceso hacia quien lo aplica. Esta estrategia de la Ingeniería de Métodos

Situacional se basa en la creación de fragmentos situacionales de método, que son luego orquestados a través de intenciones, directivas y contextos para dar coherencia al modelo de procesos que se define.

Aun cuando el argumento expuesto y la estrategia planteada pueden ser usados para la definición completa de un método que contemple todas las fases del ciclo de desarrollo de software, se acota este trabajo de investigación al Proceso de Implementación de Software que comprende lo siguiente:

- Diseño detallado
- Aprovisionamiento
- Codificación
- Verificación y validación
- Preparación de los ambientes y herramientas
- Definición de los estándares de desarrollo
- Integración de componentes.

La selección del Proceso de Implementación como caso particular de estudio se hace debido a que los métodos ágiles presentan un énfasis mayor en las actividades de Implementación de Software que los métodos orientados al plan, se considera que es justamente en este proceso de implementación donde las prácticas ágiles pueden hacer grandes aportes con menos polémica a estos métodos tradicionales.

Por lo expuesto anteriormente, se plantea la necesidad de definir los fragmentos situacionales, de acuerdo a los principios de la Ingeniería de Métodos Situacionales, que contemple prácticas extraídas y analizadas a partir de los métodos tanto ágiles como orientados al plan y que dicten la pauta al equipo de desarrollo durante la ejecución de un Proceso de Implementación de Software.

2 Objetivos de la Investigación

2.1 Objetivo General

Proponer un modelo de proceso ágil y disciplinado para la implementación de software que pueda ser ajustado a diferentes contextos de desarrollo de software, especialmente en proyectos desarrollados por las PYMES venezolanas.

Esta propuesta debe basarse en la ingeniería de métodos situacionales.

2.2 Objetivos específicos

1) Definir fragmentos de método para el proceso de implementación de software considerando, entre otras, las características contextuales de un proyecto.

2) Crear directivas de selección y ejecución que ayuden al usuario del método a determinar cuando aplicar un fragmento de método.

3 Justificación de la Investigación

Los métodos ágiles de desarrollo de software poseen dos ventajas competitivas especiales sobre sus contrapartes disciplinadas, a saber la gran adaptabilidad al cambio y a la reducción de los ciclos de retroalimentación para determinar tanto que se está construyendo el software de manera correcta (verificación) como el que se esté construyendo el software correcto (validación).

En particular, un Proceso de Implementación con características ágiles, que pueda ser integrado en un método de desarrollo orientado al plan puede incrementar la capacidad de retroalimentación para maximizar la verificación y validación de los componentes, además que permitiría a los procesos de validación posteriores acelerar la retroalimentación por parte de los clientes y usuarios finales dado que este tipo de procesos apunta a disponer de un sistema ejecutable ante cualquier cambio o incremento funcional que se realice. De igual manera, la inclusión de prácticas ágiles como la implementación de diseños simples, el desarrollo orientado a pruebas y los ajustes de diseño por refactorización permiten hacer que los cambios en el código como producto de cambios en los requerimientos sea más fácil y menos costoso de llevar a cabo.

Por otro lado, reconociendo que no todas las prácticas ágiles pueden ser llevadas a cabo en cualquier escenario, es de gran importancia incluir los lineamientos y los argumentos que soportan la toma de decisiones ante la elección de las prácticas que deben ser realizadas durante la implementación de componentes. Es de hacer notar que los argumentos para tales decisiones no pueden ser absolutos, sino que están asociados a las circunstancias y condiciones particulares de un proyecto, incluso en un momento dado.

Asimismo, siendo que un resultado deseable de la aplicación de métodos de desarrollo es la instauración de las mejores prácticas a nivel organizacional y que esto es un producto de una gestión del conocimiento de estas prácticas y de las situaciones apropiadas para su aplicación, es importante que las razones por las que las decisiones son tomadas se hagan explícitas en el modelado de procesos, transmitiendo de manera clara y directa el conocimiento subyacente tras una decisión particular.

Son estos los argumentos que justifican y validan la definición de los fragmentos de un método situacional ágil y disciplinado para la Implementación de Software.

4 Metodología

Las actividades a ejecutar para el desarrollo de esta investigación son:

1. Crear un modelo conceptual integrado a partir de las caracterizaciones del proceso de implementación tal como se observa en los métodos, modelos, estándares y guías estudiados.

2. Determinar las variables contextuales a considerar de los proyectos de desarrollo de software.

3. Analizar los puntos claves para la definición de fragmentos de método del proceso de implementación de software en el contexto de los métodos, modelos, estándares y guías estudiados y de acuerdo a las variables contextuales.

4. Generar casos base para los fragmentos de método extraídos del modelo conceptual integrado que faciliten el análisis de casos de estudio reales.

5. Analizar casos de estudio reales para la observación de las decisiones tomadas en torno a los fragmentos de método básicos que fueron obtenidos previamente de manera teórica, en el contexto de los valores específicos de las variables características del proyecto.

6. Generalizar el modelo conceptual situacional del proceso de implementación a partir de los casos de estudio observados.

7. Analizar el modelo conceptual obtenido para proponer las directivas de selección de los fragmentos de método.

8. Analizar los resultados.

9. Elaborar el documento de tesis.

10. Presentar los resultados de esta investigación

5 Resultados Esperados

Luego de la ejecución de las actividades propuestas en la sección 4, se espera que este trabajo de investigación arroje los siguientes resultados:

1. Un conjunto de fragmentos de métodos situacionales validados para el proceso de implementación de software, los cuales podrán ser adaptados a las características particulares de diversos proyectos de desarrollo.

2. Los lineamientos de instanciación y uso provistos a través de las directivas y los contextos, productos propios de una especificación situacional de fragmentos de método.

CAPITULO II: Estado del arte en Procesos de Implementación de Software

En este capítulo se describe el estado del arte en todo lo referente a Procesos de Implementación de Software, evaluando distintos modelos conceptuales asociados a este proceso en el marco de los diferentes modelos de mejoras de procesos, estándares de desarrollo de software, cuerpos de conocimientos y algunos métodos de desarrollo de software, representantes éstos de los enfoques ágil y orientado al plan.

La primera sección describe el alcance de lo que para efectos del presente trabajo de investigación, y a lo largo del mismo, se denominará Proceso de Implementación de Software. En la sección dos se hará un análisis detallado de lo que se entiende por enfoque ágil y disciplinado, y cuales son los argumentos por los que se clasifica en un enfoque u otro un método de desarrollo. Las secciones tres y cuatro se estudian modelos de mejora de procesos, estándares, guías y métodos representantes del enfoque orientado al plan y el enfoque ágil, respectivamente, procurando extraer de cada uno de ellos un modelo conceptual que permita la caracterización de los Procesos de Implementación tal como los describen allí. En la quinta sección se analizan los modelos conceptuales obtenidos de las secciones tres y cuatro, mostrando las similitudes y diferencias a la vez que se hace una propuesta de integración de los conceptos de ambos enfoques en un modelo que será usado para la generación de los fragmentos de método en los capítulos IV y V de este trabajo de investigación. Finalmente, la sección seis presenta las conclusiones de este capítulo.

1 Alcance del Proceso de Implementación en el Ciclo del Desarrollo de Software

A los términos de esta investigación, se define como Proceso de Implementación aquellas fases del desarrollo de software que van desde el diseño detallado de los componentes de un producto de software, hasta que se integran satisfactoriamente los componentes en un producto verificado y listo para las pruebas funcionales, de validación o de aceptación, según sea el caso. Así, las siguientes actividades se consideran como parte del Proceso de Implementación:

- Diseño detallado de componentes: El proceso de detallar una especificación de requisito y arquitectónica para una funcionalidad o un componente en específico hasta convertirlo en tareas que pueden ser asignadas a uno o más desarrolladores.

- Aprovisionamiento de componentes: Se refiere a las actividades necesarias para hacer disponible un componente o subcomponente que se ha determinado como necesario durante el diseño detallado. Esto puede ocurrir en forma de la creación o codificación de un nuevo componente o bajo la forma de reutilización de componentes previamente disponibles en alguna librería de activos reutilizables.

- Codificación: El proceso de escribir el código fuente producto de las tareas asignadas, que se encuentre en condición de ser exitosamente compilado. Esta actividad incluye la de prueba unitaria de los componentes codificados. En el caso de reutilización de componentes, cualquier adaptación que se requiera interna o externa al componente reutilizado también está incluida en esta actividad.

- Generación de documentos de soporte: La información de soporte (manuales de usuario, operación y mantenimiento) es principalmente producida por el equipo de desarrolladores, por lo que se hace conveniente la inclusión de esta actividad en el Proceso de Implementación.

- Integración de componentes: El proceso de hacer que los componentes, que han sido codificados o adaptados por separado, interactúen para producir las características emergentes del macrosistema que les contiene. De igual manera este proceso debe comprobar la correcta interoperabilidad entre tales componentes efectuando pruebas de integración.

- Verificación y Validación (V&V) de componentes (Parcial): Algunas actividades de la Verificación y Validación son incluidas esta definición del alcance del Proceso de Implementación, como el diseño de casos de prueba e implementación de conductores automatizados de pruebas, tanto para las pruebas unitarias de componentes como para las pruebas de integración. De igual manera se incluyen actividades que persiguen la verificación del código fuente generado, como lo son las inspecciones de código o las revisiones de pares. Quedan excluidas de este alcance, sin embargo, aquellas actividades de V&V que involucran la participación directa del usuario final o cliente, como la presentación de prototipos, demostraciones en vivo, revisiones conjuntas de especificaciones, entre otras.

En el caso del desarrollo de software por iteraciones, se considera que el Proceso de Implementación contempla una parte importante de estas iteraciones, pero no su totalidad. Algunos de los procesos y actividades llevadas a cabo en el contexto de tales iteraciones, como lo son el refinamiento de los requisitos y del diseño arquitectónico o el proceso de validación del producto resultante de la iteración, no se consideran como parte del Proceso de Implementación. De esta manera, en un modelo de desarrollo iterativo, el Proceso de Implementación se ejecutaría completo por lo menos una vez en cada iteración.

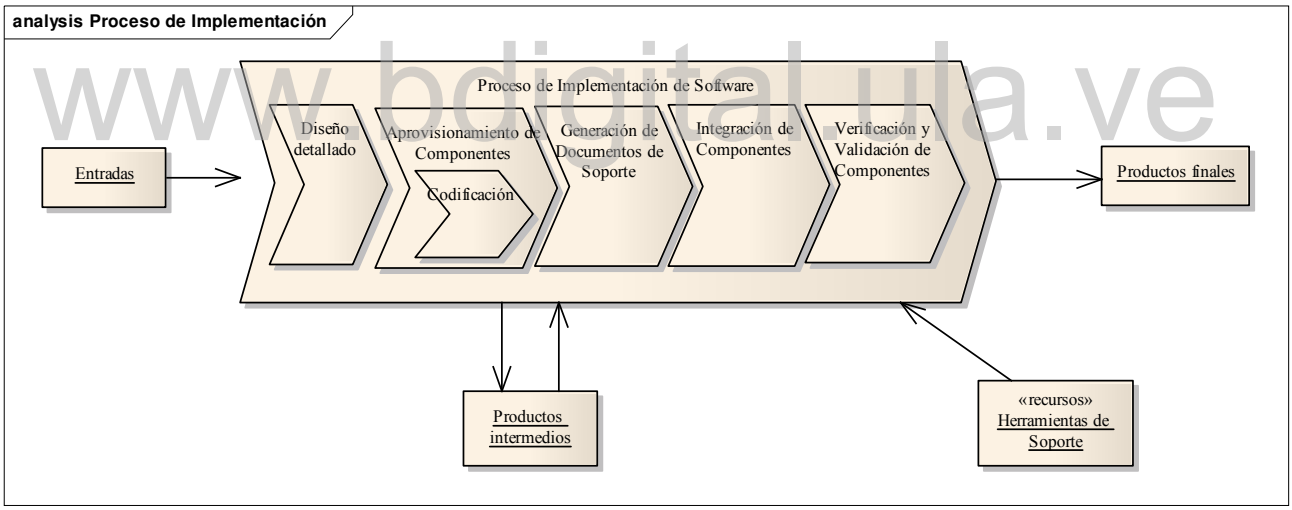


Figura 1. Diagrama del Proceso de Implementación de acuerdo al alcance establecido

En cuanto a los procesos de soporte al ciclo del desarrollo, con la excepción mencionada del proceso de V&V, tales procesos no se incluyen como parte de la implementación. Puede observarse que el patrón sugerido consiste en mantener dentro del Proceso de Implementación las actividades que resulta ineficiente asignar a algún rol separado a los desarrolladores. La figura 1 describe el Proceso de Implementación de acuerdo al alcance definido para este trabajo, mostrando también los tipos de productos contemplados en el modelo.

2 Definiciones de Disciplina y Agilidad

Un aspecto importante de la tarea que se ha propuesto en este trabajo de investigación es la clarificación asociada a los términos *Disciplina* y *Agilidad*. El término *Agilidad* está asociado a los métodos ágiles, que se

adhieren al Manifiesto Ágil (Fowler y Highsmith, 2001), y se usa indicar la capacidad de un método de ser ligero, adaptable al cambio, flexible y orientado a las personas antes que a los procesos (Fowler, 2005).

Definir la *Disciplina* en un método es un asunto polémico. Boehm y Turner explican que el término *disciplinado* puede entenderse como "conformidad con procesos y estándares establecidos" y como "auto-control", lo que ha llevado a que los partidarios de los métodos tradicionales confinen esta palabra a conformidad con procesos y estándares, mientras los partidarios de los métodos ágiles la confinan al concepto de auto-control (Boehm y Turner, 2004). Así pues, de la misma manera en que efectivamente un método de desarrollo de corte tradicional como PSP puede ser considerado como altamente disciplinado, un método ágil como Programación Extrema (*eXtreme Programming*, XP por sus siglas en inglés) es también considerado como altamente disciplinado (Ambler, 2007).

El cómo referirse a los métodos tradicionales, distintos a los métodos ágiles, también ha sido un tema difícil. Algunos autores los denominan pesados (*heavyweight*) (por ej., Balbo y Khan, 2004), otros los llaman tradicionales (McMahon, 2004), e ingenieriles (Fowler, 2005) así como Boehm y Turner tratan de presentar a estos métodos como asociados a la disciplina (Boehm y Turner, 2004). No obstante, son estos últimos autores quienes se encuentran entre los primeros en aplicar el término más usado en la actualidad para distinguir a los métodos de desarrollo de alta elaboración, en comparación con los ágiles: Orientados al Plan (*Plan-Driven*). Este trabajo de investigación se hace uso de esta denominación para los métodos tradicionales por ser la que más aceptación tiene en la actualidad.

El uso que durante este trabajo de investigación se hará de los términos *Disciplinado* y *Disciplina* hará referencia a la conjunción de las acepciones de conformidad con los procesos y estándares establecidos y auto-control.

3 Procesos de Implementación en el Desarrollo orientado al plan

La definición, y consecuente adopción y mejora de un proceso de desarrollo acorde con los contextos organizacionales y los tipos de aplicaciones a desarrollar, ha promovido la investigación tanto a nivel de prácticas y procesos estándares que deben ser implantados en las organizaciones productoras de software, como a nivel de la formación de profesionales en ingeniería de software y en el desarrollo de las competencias requeridas para realizar las actividades propias de la implementación de productos de software. Es por ello, que esta sección presenta en primera instancia el Modelo Integrado de Capacidad y Madurez (*CMMI*®), luego la guía del Cuerpo de Conocimiento de la Ingeniería de Software (*Software Engineering Body of Knowledge*, *SWEBOK* por sus siglas en inglés). Asimismo, se presentan los estándares ISO/IEC 12207 e *IEEE* 1074, como los más utilizados para definir modelos de procesos de desarrollo de software, y sobre los cuales se fundamentan gran parte de los métodos de desarrollo publicados (ISO/IEC JTC 1, 2000, *IEEE* Standards Board, 1998). Al final de la sección se analiza el método *Gray Watch* (Montilva, Barrios y Rivero, 2008) como representante de los métodos de desarrollo de software dentro de la categoría disciplinada.

3.1 Modelo Integrado de Capacidad y Madurez (*CMMI*®)

Según (*CMMI Product Team*, 2006), el Modelo Integrado de Capacidad y Madurez (*CMMI*® por sus siglas en inglés) es un enfoque de mejora de procesos que provee elementos esenciales de procesos efectivos para una o más disciplinas, con el fin de ayudar a las organizaciones a mejorar sus procesos de desarrollo y mantenimiento de productos y servicios relacionados con tecnologías de información.

El modelo *CMMI* fue desarrollado por un equipo de producto conjunto formado por representantes de la industria, el gobierno de los EEUU y el Instituto de Ingeniería de Software (SEI) de la Carnegie Mellon University, se basa en el *Framework CMMI* (Bate y Shrum, 1998) el cual agrupa las mejores prácticas asociadas a diferentes áreas de interés, denominadas *Constelaciones*. Hasta el momento se han desarrollado tres (3) constelaciones distintas: *CMMI*® para Desarrollo (*CMMI-DEV*® por sus siglas en inglés) en el 2006, *CMMI*® para Adquisición (*CMMI-ACQ*® por sus siglas en inglés) en el 2007 y *CMMI*® para Servicios (*CMMI-SVC*® por sus siglas en inglés) en el 2009. En todo este trabajo de investigación se usarán indistintamente los términos *CMMI*® y *CMMI-DEV*® para referirse a la última versión de la *constelación CMMI*® para Desarrollo versión 1.2 (*CMMI Product Team*, 2006).

CMMI® está basado en la premisa de que la calidad de los productos está fuertemente influenciada por la calidad de los procesos usados para su desarrollo y mantenimiento. Por ello, este modelo establece un estándar alto para la evaluación de la calidad de procesos, lo que le ha dado la reputación de ser el principal exponente del enfoque tradicional y orientado al plan, razón suficiente para su inclusión en este trabajo de investigación.

3.1.1 Estructura conceptual de los Modelos Integrados de Capacidad y Madurez (*CMMI*®)

El modelo *CMMI* está compuesto por áreas de procesos, metas específicas, metas genéricas, prácticas específicas y prácticas genéricas. Un área de procesos es un grupo de prácticas relacionadas entre sí que, cuando son implementadas de manera colectiva, satisfacen un conjunto de metas que se consideran como importantes para poder hacer mejoras en los procesos asociados. Las metas específicas describen las características únicas que deben estar presentes para satisfacer un área de procesos. Por su parte las metas genéricas, llamadas así porque todas ellas aplican a múltiples áreas de procesos, describen las características que deben estar presentes para institucionalizar los procesos implementados en un área de procesos. Las prácticas específicas describen las actividades que se consideran importantes para lograr la meta específica asociada. Las prácticas genéricas describen a su vez las actividades consideradas importantes para lograr las metas genéricas. El modelo *CMMI*® para Desarrollo v1.2 esta compuesto de 22 áreas de procesos.

El modelo tiene dos representaciones distintas para la evaluación de la capacidad y la madurez de los procesos de una organización: Representación Escalonada (por niveles) y Representación Continua. La representación escalonada ofrece una manera sistemática y estructurada de aproximarse a la mejora de procesos basada en modelos un escalón o nivel a la vez. El modelo establece 5 niveles de para clasificar la madurez de los procesos organizacionales, en función de las áreas de procesos que alcanzan sus objetivos y pueden ser gestionadas con principios de ingeniería. Por su parte, la representación continua permite a una organización mejorar diferentes áreas de procesos cada una a su paso, logrando así la mayor flexibilidad al momento de seleccionar las áreas de procesos que serán mejoradas

3.1.2 Proceso de implementación en el contexto de *CMMI*®

Las actividades descritas en la sección 1 como asociadas al Proceso de Implementación de Software, se encuentran enmarcadas en el modelo *CMMI*® en las áreas de proceso *Solución Técnica*, *Integración de Productos* y *Verificación*.

El diseño y la construcción de software está incluida en el área de procesos *Solución Técnica*, más específicamente, se corresponde con las metas *SG2: Desarrollar el diseño* y *SG3: Implementar el diseño del producto*. Esta área de procesos tiene como propósitos diseñar, desarrollar e implementar soluciones a los requisitos del sistema y se enfoca en evaluar y seleccionar soluciones que potencialmente satisfagan un

conjunto de requisitos establecidos y desarrollar diseños detallados para tales soluciones. El nivel de detalle requerido incluye toda la información necesaria para precisar, codificar e implementar el diseño como un producto o un componente de producto.

La meta específica *SG2* de desarrollo del diseño está asociada al proceso de Construcción cuando se trata del diseño de componentes. Este proceso de acuerdo con el contexto planteado por *CMMI* está relacionado con la conformación de un Paquete de Datos Técnicos, el cual proporciona la descripción del componente de producto y que incluye todos los datos técnicos tales como dibujos, listas asociadas, especificaciones, descripciones de diseño, bases de datos de diseño, estándares, requisitos de desempeño (*performance*), provisiones de aseguramiento de la calidad y detalles de empaque del componente. Entre las prácticas asociadas al diseño detallado se contempla la selección de criterios de evaluación de la calidad del diseño, la revisión constante del diseño y la adhesión a estándares existentes.

La meta específica *SG3* de implementación del diseño contiene las prácticas específicas de ejecución y de documentación del proceso de implementación. Como sub-prácticas en la implementación del diseño, se plantea el uso de métodos efectivos para dicha implementación, tales como programación estructurada, programación orientada a objetos, generación automática de código reutilización de código y el uso de patrones de diseño aplicables; la adhesión a criterios y estándares aplicables a la implementación, la realización de revisiones de pares y de pruebas unitarias de los componentes implementados. La práctica de documentación contempla todos los documentos que son necesarios para la instalación, operación y mantenimiento del producto, como manuales de usuario, manuales de entrenamiento, manual de operación, manual de mantenimiento y la ayuda en línea.

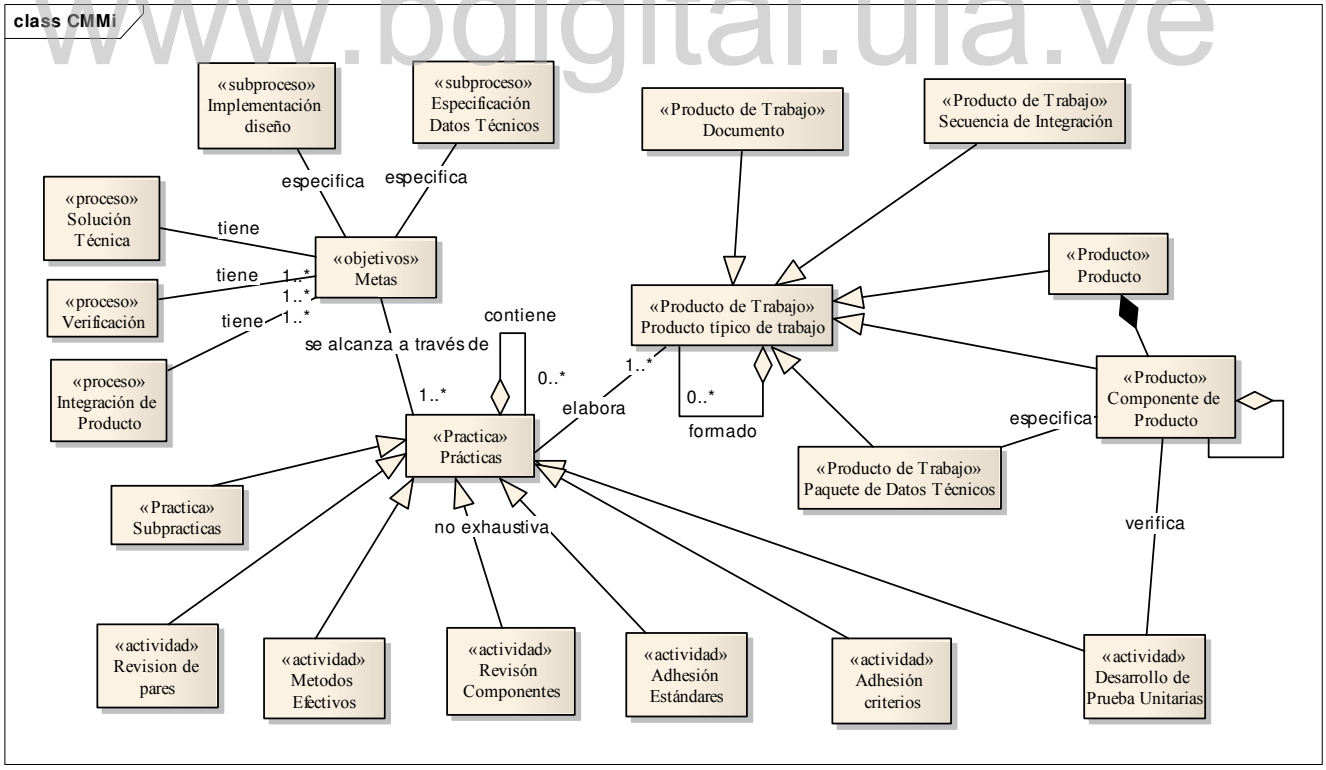


Figura 2. Modelo Conceptual del Proceso de Implementación de Software según el modelo *CMMI*

El área de procesos de *Verificación*, también es importante para el estudio que se lleva a cabo en este trabajo, puesto que contempla las metas y las prácticas necesarias para asegurar que los productos de trabajo y los componentes de producto cumplen con los requisitos establecidos y especificados. Para lograr este objetivo, se utiliza la verificación formal de los requisitos, la cual es vista como un proceso que requiere ser planeado, ejecutado y analizado generando sus correspondientes incidencias y métricas y la realización de revisiones de pares en las actividades propias de la construcción del producto de software.

En esta área de procesos de *Verificación,* el modelo *CMMI*® da particular importancia a la técnica de revisión de pares como proceso fundamental de verificación, lo que se demuestra en la meta *SG2: Desarrollar revisión de pares.* Sin embargo, otras técnicas de verificación pueden ser incluidas, planificadas y ejecutadas en las metas *SG1: Preparación para la verificación* y *SG3: Verificar los productos de trabajo seleccionados*.

Las actividades asociadas a la integración de componentes se encuentran enmarcadas, por su parte, en el área de procesos *Integración de Productos*: Las metas de esta área son: (1) *SG1: Preparación para la integración de producto*, donde se debe determinar la secuencia de integración; establecer el ambiente de integración; y, establecer los procedimientos y los criterios de integración de producto; (2) *SG2: Asegurar la compatibilidad de las interfaces*, que consiste en revisar las descripciones de las interfaces para determinar completitud y gestionar las interfaces; y *(3) SG3: Ensamblar los componentes de producto y entregar el producto*, que permite confirmar la preparación de los componentes de producto para la integración; ensamblar los componentes de producto; evaluar los componentes de producto ensamblados; empacar y entregar el producto o el componente de producto. En la figura 2 se presenta un modelo conceptual derivado del análisis del proceso de Implementación de Software desde la perspectiva del modelo *CMMI*.

3.2 Guía del Cuerpo de Conocimientos de la Ingeniería de Software (*SWEBOK*®)

La Guía del Cuerpo de Conocimientos de la Ingeniería de Software (*Guide to the Software Engineering Body of Knowledge*, *SWEBOK*® por sus siglas en inglés) es una representación curricular que detalla el subconjunto de conocimiento generalmente aceptado como requerido para ejercer la profesión de la Ingeniería de Software (*IEEE Computer Society*, 2008). El proyecto *SWEBOK*® ha tenido como intención el alcanzar el consenso en la definición y aceptación de un cuerpo central de conocimientos, puesto que este consenso es visto como crucial y un hito clave en la evolución de la Ingeniería de Software en su estatus como profesión. (Abran et al, 2004).

La primera versión de la Guía del *SWEBOK*® se hizo disponible en el año 2001 (SWEBOK® 2001), mientras que la más reciente es la versión *SWEBOK*® 2004, publicada como su nombre lo indica en el año 2004. En la actualidad, se espera que para el 2010 se publique la nueva versión denominada *SWEBOK*® 2010. En todo momento, al hacer mención de la Guía del *SWEBOK* se hará referencia a la última versión disponible (SWEBOK® 2004).

El carácter propiciador del consenso en el cuerpo de conocimientos que persigue la Guía del *SWEBOK*®, así como la definición del conocimiento básico que debe presentar todo profesional que desee ejercer la Ingeniería de Software hacen de esta guía portador de un modelo conceptual interesante para el objetivo que se propone este trabajo de investigación, razón por la que se incluye en esta revisión del estado del arte en Procesos de Implementación.

Es importante resaltar que la última versión de la guía del *SWEBOK*® incluye menciones tanto a métodos y estándares orientados al plan como a métodos ágiles y a la selección de métodos en el eje Orientación al Plan

– Agilidad (Abran et al, 2004), no obstante, las prácticas mencionadas como parte del Proceso de Implementación sólo se limitan a aquellas en las que existe consenso en la comunidad (por ejemplo Desarrollo guiado por pruebas y Refactorización del diseño), sin hacer mención de aquellas prácticas ágiles en las que no existen consenso, como Integración Continua o Programación por Pares, razón por la que la se enumeran entre los modelos y métodos asociados al desarrollo tradicional y orientado al plan.

3.2.1 Estructura conceptual de la Guía del Cuerpo de Conocimientos de la Ingeniería de Software

La Guía del *SWEBOK* se encuentra organizada en diez (10) áreas de conocimientos, de las cuales las primeras cinco son las que ISO reconoce en el estándar 12207 como los procesos primarios, mientras que las cinco restantes representan los procesos de soporte y organizacionales de la Ingeniería de Software. Estas áreas de conocimiento son: Requerimientos de Software, Diseño de Software, Construcción de Software, Pruebas de Software, Mantenimiento de Software, Gestión de la Configuración del Software, Gestión de la Ingeniería del Software, Proceso de Ingeniería de Software, Métodos y Herramientas de Ingeniería de Software y Calidad del Software.

Cada una de estas áreas del conocimiento, que representa un capítulo de la Guía del *SWEBOK*, es descompuesta en un conjunto de tópicos organizados en dos o tres niveles de profundidad, procurando que tal subdivisión sea compatible con aquellas divisiones usadas por la industria, por la literatura disponible y los estándares existentes. Estas secciones son presentadas como sub-áreas, temas y sub-temas. Finalmente, por cada área de conocimiento se presenta una matriz con los temas y sub-temas tratados y un extracto de las fuentes referenciales disponibles donde puede profundizarse en cada uno de ellos, así como una lista de referencias recomendadas para una lectura posterior referente al área de conocimiento y sus temas asociados.

3.2.2 Proceso de Implementación de Software en el contexto de *SWEBOK*

El Proceso de Implementación de Software aparece enmarcado dentro de la Guía del *SWEBOK* como una de las diez áreas de conocimiento denominada *Construcción de Software*. La guía define la construcción de software como “la creación detallada de software funcional y significativo por medio de la combinación de codificación, verificación, pruebas unitarias, pruebas de integración y depuración del código” (Abran et al, 2001). A pesar de que esta área de conocimientos aparece asociada con todas las otras áreas de conocimiento, ésta tiene una relación mucho más estrecha con las áreas de conocimiento de Diseño y Pruebas de Software. Esto se debe, principalmente, a que el proceso de construcción de software involucra actividades significativas del diseño y de las pruebas. El área de Construcción de Software tiene tres (3) sub-áreas: *Fundamentos de la Construcción*, *Gestión de la Construcción* y *Consideraciones Prácticas.*

La sub-área de *Fundamentos de la Construcción* prescribe las directrices que deben seguirse al desarrollar actividades dentro de este proceso. Tales fundamentos son:

- minimización de la complejidad, enfatizando la creación de código simple y legible antes que astuto;
- anticipación del cambio, orientando el proceso de construcción hacia la preparación y facilitación de cambios posteriores en el código fuente debido a cambios en el entorno del software;
- construcción para la verificación, mediante la inclusión de prácticas que permitan cotejar con facilidad, manual o automáticamente, que el producto creado cumple con la especificación provista.

Por su parte, la sub-área de Gestión de la Construcción contempla las actividades de preparación y soporte del proceso de la construcción, así como la captura de métricas asociadas a las distintas actividades de construcción, lo que permite una revisión *a posteriori* del desempeño del proceso de construcción, y su consecuente, mejora en función de la evolución de dichas métricas a lo largo del tiempo.

Finalmente, la sub-área de *Consideraciones Prácticas* menciona las prácticas asociadas al proceso de Construcción de Software. La Guía del *SWEBOK*® describe a este proceso como uno con un carácter eminentemente pragmático. Contiene las siguientes consideraciones:

- toda construcción involucra algún tipo de diseño detallado; incluso debe contemplarse el que el programador durante la codificación deba realizar modificaciones que superen las brechas no cubiertas durante la fase de diseño;

- durante la actividad de codificación, debe procurarse la legibilidad del código, el uso de estructuras de control y datos, el manejo de errores y excepciones, la consistencia en la organización del código fuente, su documentación y su entonación; aplicación de pruebas unitarias y de integración durante la construcción del software;

- formalizar la práctica de reutilización al integrar procesos en el ciclo de vida del software; y, contemplar la aplicación de técnicas de aseguramiento de la calidad en la construcción del software, como el desarrollo guiado por pruebas, las revisiones técnicas, el uso de aserciones en el código, las técnicas de depuración y el análisis estático;

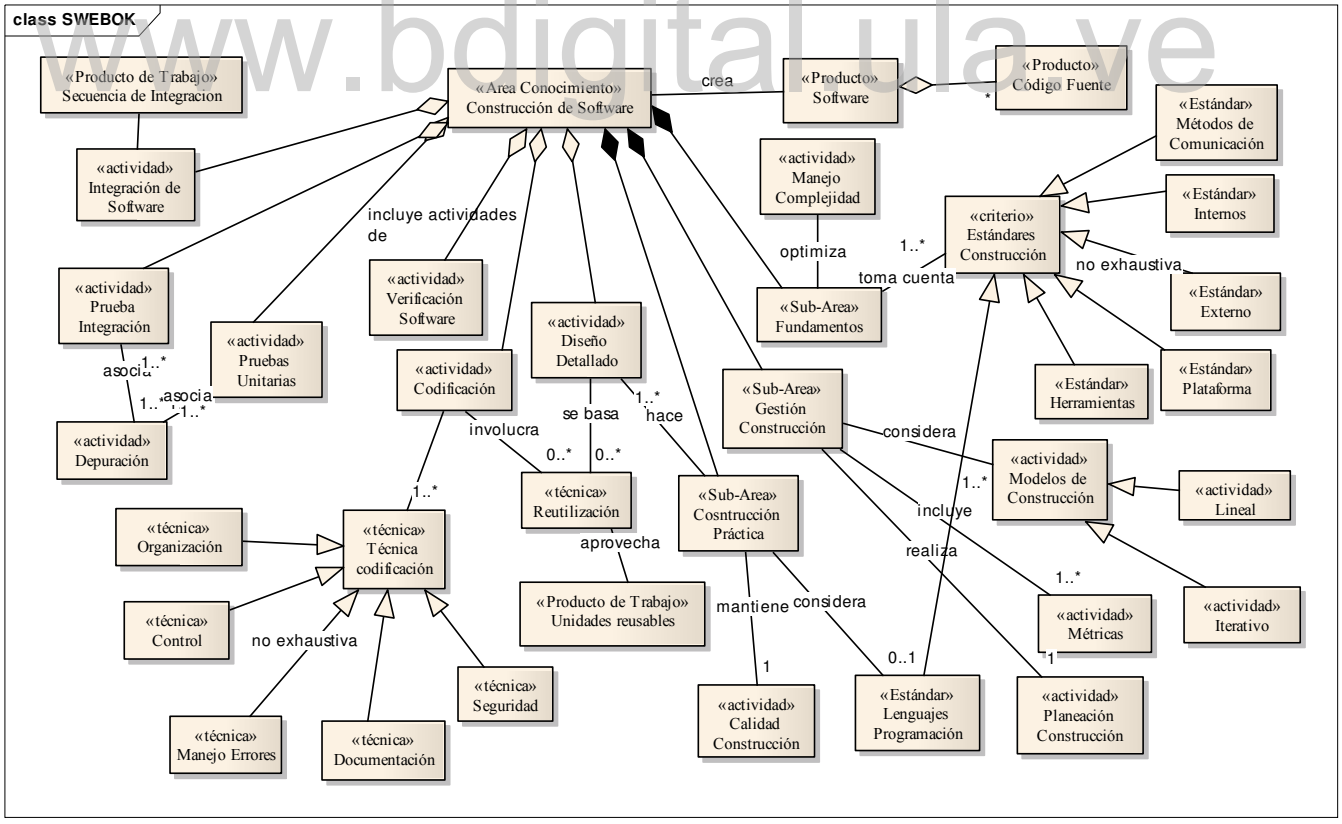


Figura 3. Modelo Conceptual del Proceso de Implementación de Software según la Guía del *SWEBOK*

- los componentes construidos de manera separada deben ser integrados entre si, o bien, con otros sistemas hardware o software, siguiendo una secuencia de integración previamente planeada y contemplando la realización de pruebas de integración que verifiquen los componentes antes y después de haber sido integrados.

La figura 3 muestra los conceptos involucrados en la construcción de software según la Guía *SWEBOK*.

3.3 Estándar ISO/IEC 12207

De acuerdo con (ISO, 2008), el estándar ISO/IEC 12207 establece un marco común de trabajo para los procesos del ciclo de vida del software que pueden usarse como referencia en la industria del desarrollo de software. En este estándar se consideran los procesos, actividades y tareas que deben ser aplicadas durante la adquisición de un producto o servicio de software y durante el suministro, desarrollo, operación, mantenimiento y disposición de productos de software.

La primera versión del estándar ISO/IEC 12207 fue desarrollado por la un Comité Técnico Conjunto formado por la Organización Mundial para la Estandarización (ISO) y la Comisión Internacional Electrotécnica (*International Electrotechnical Commission*, IEC por sus siglas en inglés) en 1995. Este estándar cubre la necesidad crítica de un marco de trabajo uniforme para la gestión e ingeniería del software, lo que promueve el comercio internacional en productos y servicios de software.

3.3.1 Estructura conceptual del Estándar ISO/IEC 12207

La estructura del estándar se basa en una arquitectura de basada en un conjunto de procesos y las interrelaciones de esos procesos. Este estándar se basa en dos principios fundamentales para la definición de los procesos: *Modularidad*: Los procesos son modulares, con un mínimo acoplamiento y una máxima cohesión, y donde cada proceso se dedica a realizar una única función; y *Responsabilidad:* Cada proceso debe tener un responsable de su ejecución, facilitando la aplicación del estándar en proyectos en los que pueden existir distintas personas u organizaciones involucradas.

Los procesos se clasifican en tres tipos: *Principales*, de *Soporte* y *Organizacionales*. Los procesos de soporte y organizacionales deben existir independientemente de la organización y del proyecto ejecutado. Los procesos principales se refieren a aquellos procesos que son los motivadores primarios en el ciclo de vida (adquisición, aprovisionamiento, desarrollo, operación y mantenimiento) y se instancian de acuerdo con la situación particular. Debido a lo amplio del alcance de este estándar, los procesos principales engloban todas las actividades del ciclo de desarrollo de software, desde el análisis de los requerimientos hasta el despliegue de la aplicación.

3.3.2 Proceso de Implementación de Software en el contexto del Estándar ISO/IEC 12207

Tal como se mencionó anteriormente, dentro del grupo de procesos principales se encuentra el proceso de *Desarrollo*, el cual a su vez contiene las actividades de diseño, de codificación, pruebas e integración del producto de software. Así, las actividades que se relacionan con los procesos de construcción, verificación e integración de software correspondiente al proceso de implementación establecido en este trabajo son las siguientes:

- Diseño detallado del Software: Esto es referido como el proceso de refinación del diseño de los componentes del software a niveles más bajos para la codificación, compilación y pruebas del software. Entre

otras cosas, los siguientes componentes de software deben ser explícitamente diseñados con suficiente nivel de detalle por el desarrollador: Interfaces externas, Diseño detallado de cada componente, Base de Datos, Requerimientos de las Pruebas de Integración.

- Codificación y pruebas del software: Se plantea la necesidad de la creación simultánea del código, la base de datos, los datos y procedimientos de pruebas unitarias, de verificación y validación de componentes, así como la documentación de todos los anteriores.

- Integración del Software: Contempla la creación de un plan de integración sobre las unidades y componentes de software y la generación de los procedimientos y elementos para la realización de las pruebas de integración. En la figura 4 se presenta un resumen de los conceptos manejados por este estándar.

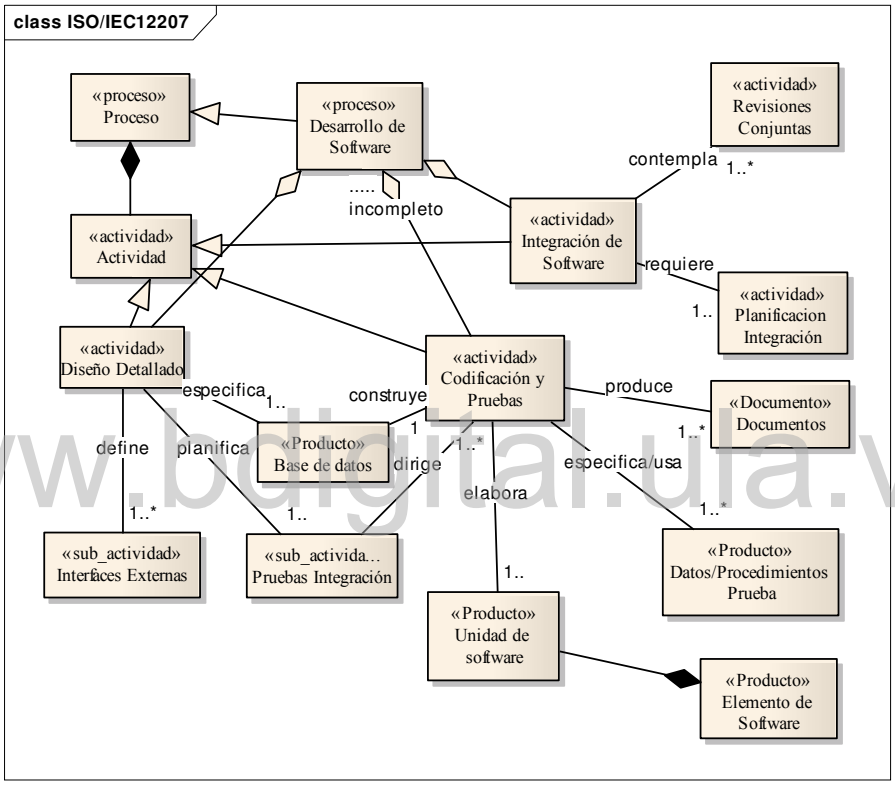


Figura 4. Modelo conceptual del Proceso de Implementación de Software según el estándar ISO/IEC 12207

3.4 Estándar *IEEE* 1074

El estándar *IEEE* 1074 (*IEEE* Standards Board, 1998) describe el conjunto de actividades y procesos obligatorios para el desarrollo y mantenimiento del software, a la vez que establece un marco común para el desarrollo de modelos de ciclo de vida y métodos de desarrollo. Contempla la generación del proceso que gobierna el desarrollo de software y el mantenimiento en un proyecto específico.

El propósito principal de este estándar es proporcionar un estándar metodológico para diseñar modelos y métodos. De allí que antes de poder usar este estándar en algún proyecto de software, primero debe diseñarse el método de desarrollo. Como puede observarse en (*IEEE* Standards Board 1998), el método enmarcado en este estándar comienza con la selección de un modelo apropiado de ciclo de vida del software para ser usado

en el proyecto en específico. Continúa con la creación del ciclo de vida del software a partir de la selección del modelo y de las actividades contempladas en el mencionado estándar.

3.4.1 Estructura conceptual del Estándar *IEEE* 1074

Este estándar divide el conjunto de actividades entre procesos y los procesos entre grupos de procesos. Los componentes del proceso de ciclo de vida del software de acuerdo con este estándar consisten de 65 actividades, organizadas en 17 grupos de actividades. Estos grupos de actividades se encuentran igualmente distribuidos en cinco secciones entre las que se tiene: a) Gestión del proyecto, b) Pre-Desarrollo, c) Desarrollo, d) Post-Desarrollo y e) Integrales.

En el contexto del estándar *IEEE* 1074, una actividad está definida como el cuerpo de trabajo que debe ser realizado, incluyendo su entrada requerida y la salida producida.

Estas actividades están conformadas por tres partes: a) Información de Entrada, que es una lista de la información requerida para ser transformada y su fuente; b) Descripción, que plantea una discusión de las acciones de valor agregado a ser ejecutadas para lograr la transformación de la entrada a la salida, y; c) Información de Salida, una lista de la información que se requiere sea generada por la transformación y su destinatario.

3.4.2 Proceso de Implementación de Software en el contexto del Estándar *IEEE* 1074

Los procesos del ciclo de vida del software, tal como los presenta el estándar *IEEE* 1074, enmarcan los procesos de construcción, pruebas e integración de software en los siguientes grupos de actividades y sub-actividades:

- Diseño
    - Realizar diseño detallado: involucra la selección de las alternativas de diseño para implementar las funciones específicas a cada componente de software. Como salida de esta actividad se tiene la estructura de datos, el algoritmo y la especificación de la información de control de cada componente de software.
- Construcción
    - Crear código ejecutable: incluye no sólo el código fuente funcional que se puede compilar, sino también los comentarios incluidos en el mismo código.
    - Crear documentación operativa: la documentación operativa se refiere a aquella que es necesaria para la instalación, operación y soporte de la aplicación a lo largo de todo el ciclo de vida.
    - Realizar la integración del software: consiste en componer e integrar los distintos componentes que puedan conformar el software por separado para que conformen un solo producto entregable.
- Evaluación
    - Conducir revisiones: tanto los diseños, como las implementaciones, documentación y la integración del producto requiere revisiones constantes. Estas revisiones pueden ser llevadas a cabo durante la realización de las distintas tareas por parte de los miembros del equipo, sobre

los indicadores de gestión, sobre el proceso ejecutado y posterior a la ejecución de las actividades sobre los productos de trabajo.

En la figura 5, se muestra el conjunto de conceptos básicos del estándar *IEEE* 1074.

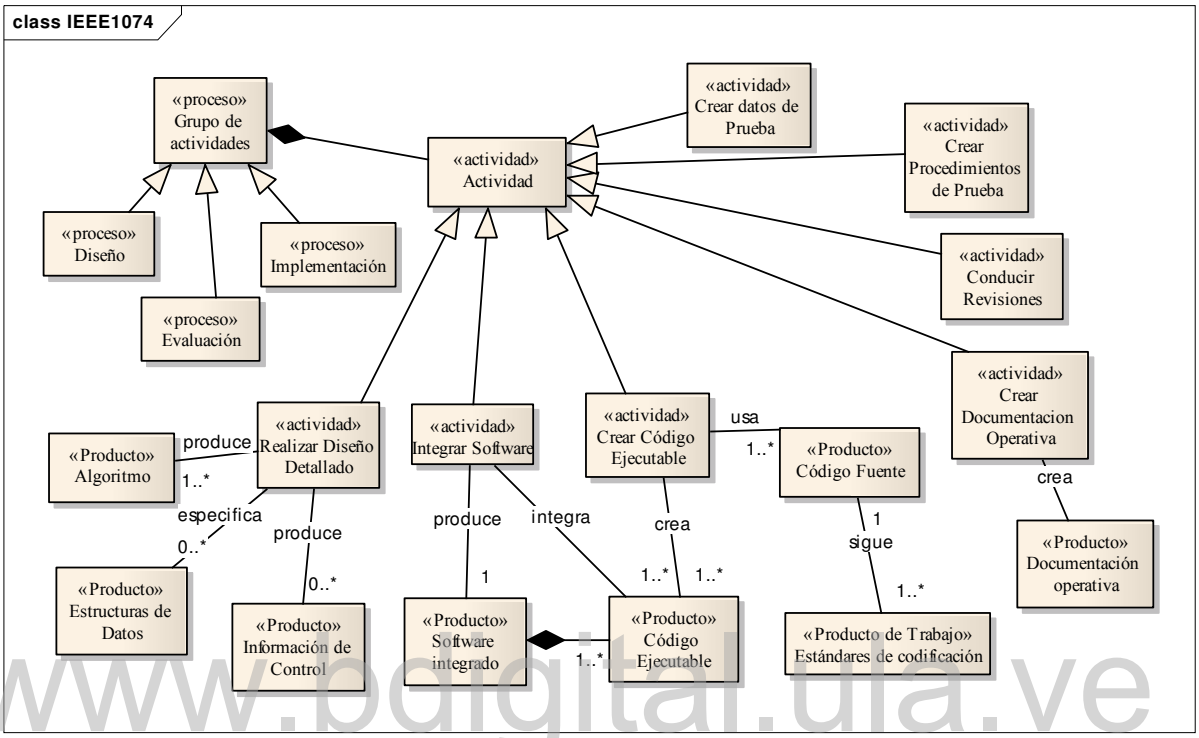


Figura 5. Modelo conceptual del Proceso de Implementación según el estándar *IEEE* 1074

3.5 Método *GRAY WATCH*

El método *WATCH* es "un marco metodológico que describe los procesos técnicos, gerenciales y de soporte que deben emplear los equipos de trabajo que tendrán a su cargo el desarrollo de aplicaciones" (Montilva, Barrios y Rivero, 2008). Mientras la primera versión publicada de este método en el año 2000 estaba orientada a proyectos pequeños y medianos en organizaciones pequeñas (Montilva, Hamzam y Gharawi, 2000), la versión Gray se encuentra orientada al desarrollo de aplicaciones empresariales (Montilva, Barrios y Rivero, 2008).

La definición de esté método de desarrollo involucró el uso como marcos de referencia de modelos como el modelo *CMMI*®, la Guía del *SWEBOK*®, el Cuerpo de Conocimientos de la Gestión de Proyectos (*Project Management Body of Knowledge*, *PMBOK*® por sus siglas en inglés), estándares como los de la Sociedad de Computación de la *IEEE* y las mejores prácticas la ingeniería de software como el desarrollo guiado por modelos, desarrollo guiado por pruebas, desarrollo iterativo, entre muchas otras.

Entre las características más resaltantes del método se contemplan las siguientes:

- Está solidamente fundamentado en los conceptos que se derivan de la Ingeniería de Software y los Sistemas de Información Empresarial.

- Es estructurado y modular.
- Tiene el propósito específico de dar soporte al desarrollo de aplicaciones de software de entornos empresariales.
- Es flexible y adaptable, con relativa facilidad, a otros tipos de productos de software.
- Emplea las mejores prácticas del desarrollo de software, aquellas internacionalmente aceptadas y mayormente utilizadas en la industria.
- Emplea las mejores prácticas y procesos de gestión de proyectos, como las establecidas en el *PMBOK*® propuesto por *PMI*.
- Integra los procesos de gestión con los procesos técnicos y de soporte.

3.5.1 Estructura Conceptual del Método *WATCH*

El método consta de tres componentes:

- **Modelo del producto**: En este modelo se identifican y describen los tipos de productos, tanto los intermedios como los finales, que se generan durante el desarrollo de una aplicación empresarial. Esto incluye tanto los documentos, modelos, listas de chequeo, librerías de software que no necesariamente forman parte de la aplicación a entregar, como aquellos productos que si integran la aplicación, como programas, bases de datos y manuales.

- **Modelo de actores**: Este modelo describe como el grupo de desarrollo debe estar organizado y cuales son los roles de cada uno de sus miembros. Estos roles se corresponden con las fases generales del desarrollo de la aplicación. Entre los objetivos de este modelo se tiene el identificar los actores o interesados que están involucrados en el desarrollo, describir las modalidades de organización del equipo de trabajo y definir los roles y responsabilidades de los actores.
- **Modelo del proceso**: Es una descripción estructurada del conjunto de actividades que el grupo de desarrollo deberá seguir para producir una aplicación empresarial. Describe un conjunto estructurado de actividades necesarias para producir una aplicación empresarial. Este modelo organiza dentro de dos tipos de procesos diferentes pero complementarios los procesos gerenciales y los procesos de desarrollo.
    - Los procesos gerenciales describen las actividades que la gerencia del proyecto (ó, en su defecto, el líder del proyecto) debe realizar para:
        - Planificar, organizar, dirigir, manejar el grupo de desarrollo y controlar el proyecto de desarrollo de un sistema o aplicación empresarial
        - Asegurar la calidad del sistema.
        - Gestionar la configuración del sistema
        - Adiestrar el grupo de desarrollo durante el proceso de ejecución del proyecto.

- Los procesos de desarrollo son los procesos técnicos que describen que debe hacer el grupo de desarrollo para producir una aplicación empresarial. Estos procesos se organizan en una estructura jerárquica formada por fases, pasos y actividades.

3.5.2 Proceso de Implementación en el contexto del Método *WATCH*

El método contempla dentro del grupo de Procesos de Implementación, los procesos de Programación & Integración y el de Pruebas de la Aplicación.

El proceso de Programación & Integración se encarga de producir, probar e integrar los componentes arquitectónicos de la aplicación, en cada una de sus versiones y consiste en: (1) elaborar, codificar y/o adaptar cada uno de los componentes que integran las diferentes versiones de la aplicación; (2) probar cada componente como una unidad; (3) integrar estos componentes de acuerdo a la arquitectura diseñada; y (4) probar la integración de estos componentes (Montilva et al, 2008).

El proceso de Pruebas de la Aplicación verifica y valida la aplicación para asegurarse que cumple con los requisitos especificados y satisface las necesidades de información y automatización que tienen sus usuarios. El proceso consiste en verificar cada versión de la aplicación como un todo y depurar los errores encontrados, a fin de asegurar que ella cumple con todos los requisitos especificados en el Documento de Requisitos. Las pruebas se realizan a tres niveles: (1) Nivel de unidad, en el cual cada componente de software es probado separadamente; (2) Nivel de integración, en el cual se prueba la integración de los componentes y sus interacciones; y (3) Nivel del sistema, en el cual la versión de la aplicación se prueba como un todo. Las pruebas de unidad y de integración tienen lugar durante el proceso de Programación & Integración mientras que las pruebas de sistema se realizan en el proceso de Pruebas de la Aplicación.

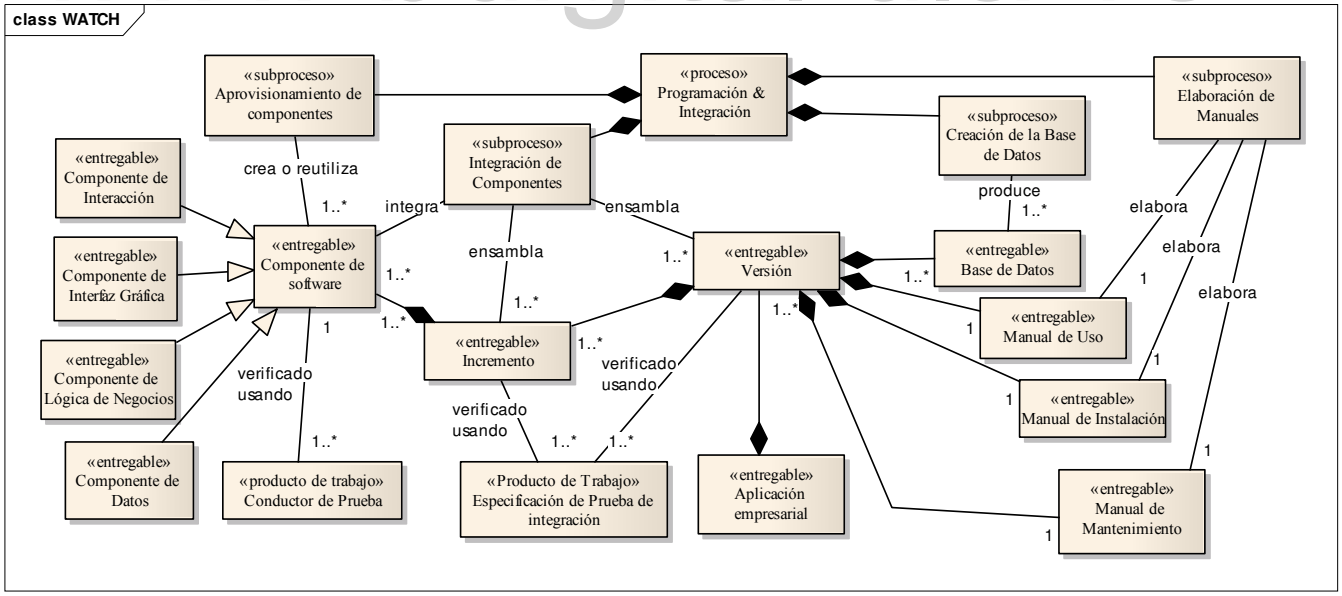


Figura 6. Modelo conceptual del Proceso de Implementación en el contexto del Método WATCH

En el grupo de Procesos de Implementación se producen los siguientes productos técnicos intermedios y/o finales: Especificaciones de Pruebas, Mecanismos de Pruebas, Componentes de Software, Incrementos, Bases de Datos, Manual de Instalación, Manual de Uso, Manual de Mantenimiento, Versión de la aplicación, Aplicación

Empresarial finalizada. Se tienen entre las prácticas propuestas en los procesos asociados a la implementación de software las siguientes:

- Uso de documentos de diseño detallado que deben ser actualizados por los programadores, así como los respectivos modelos de diseño que puedan estar disponibles.

- Programación guiada por Pruebas (*Test-Driven Development*), donde las pruebas son desarrolladas previamente a la codificación de los nuevos componentes.

- La reutilización de software como principio guía para el aprovisionamiento de los componentes que se utilizarán en la construcción de la aplicación, aumentando con esto la calidad del software y llevando al mínimo el tiempo de codificación de nuevos componentes.

- Integración Incremental de los Componentes y de los incrementos, previo a la generación de versiones.

- Inclusión de Revisiones Técnicas, Análisis de la Trazabilidad y Pruebas del Software como procesos técnicos de verificación de los productos finales e intermedios.

En la figura 6 se representan los conceptos manejados por el método *WATCH*.

4 Procesos de Implementación en el Desarrollo ágil

Ambler (Ambler, 2005a) presenta una definición disciplinada del Desarrollo Ágil de Software como “un enfoque iterativo e incremental (evolutivo) para el desarrollo de software. llevado a cabo de una manera altamente colaborativa por equipos autoorganizados en un marco de trabajo de gobernabilidad efectiva, haciendo uso sólo de la ceremonia suficiente, que produce software de alta calidad de manera costo-efectiva satisfaciendo las necesidades cambiantes de sus interesados”.

Los métodos ágiles, de acuerdo a (Fowler, 2005), se desarrollaron como una reacción a los métodos de desarrollo orientados al plan, en especial atendiendo a la principal crítica que se hace de estos métodos tradicionales de que son burocráticos y que hay tanto que hacer para aplicar uno de estos métodos que sólo puede lograr ralentizar el ritmo de desarrollo. Por ello, los métodos ágiles intentan buscar un punto de balance entre ningún proceso y demasiado proceso, aportando sólo el proceso necesario.

En Febrero del 2001, un grupo de personas representantes de Programación eXtrema, *SCRUM, DSDM, Adaptative Software Development, Crystal, FDD, Pragmatic Programming* se reunieron para compartir y discutir acerca de la necesidad de una alternativa a los procesos de desarrollo de software pesado y guiado por documentación (Fowler y Highsmith, 2001). Estos pensadores, de quienes un subconjunto posteriormente se constituyó en *The Agile Alliance*, propusieron que las empresas, para ser exitosas en la nueva economía y moverse agresivamente en la era del comercio electrónico, del negocio electrónico y de la web, deberían librarse de sus maneras retrógradas de trabajar y de sus políticas anacrónicas.

El Manifiesto Ágil fue el principal resultado de la mencionada reunión (Fowler y Highsmith, 2001) y, aunque tiene más una connotación filosófica que técnica o metodológica, sirve de principio guía para la definición de métodos ágiles existentes y para determinar cuando un método puede ser efectivamente considerado como ágil. El manifiesto expresa textualmente:

"Manifiesto por el Desarrollo Ágil de Software

Estamos descubriendo mejores maneras de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de esta experiencia hemos aprendido a valorar

Individuos e interacciones *sobre procesos y herramientas*

Software que funciona *sobre documentación exhaustiva*

Colaboración con el cliente *sobre negociación de contratos*

Responder ante el cambio *sobre seguimiento de un plan*

Esto es, aunque los elementos a la derecha tienen valor, nosotros valoramos por encima de ellos los que están a la izquierda."

Tomado de (*Agile Spain*, 2005)

Este Manifiesto Ágil establece doce principios para estas *mejores maneras* de desarrollar software:

1. La mayor prioridad la tiene la satisfacción al cliente a través de la entrega temprana y continua de software valioso.

2. Se da la bienvenida a los requisitos cambiantes, incluso tarde en el desarrollo del software. Los procesos ágiles se ciñen al cambio a favor de la ventaja competitiva del cliente.

3. Entregar software que funciona frecuentemente, desde un par de semanas hasta un par de meses, prefiriendo las escalas de tiempo más cortas.

4. Las personas del negocio y los desarrolladores deben trabajar juntos todos los días durante todo el proyecto.

5. Construir proyectos alrededor de individuos motivados, dándoles el entorno y el apoyo que necesitan y confiando que ellos realizarán la tarea.

6. El método más eficiente y efectivo de transmitir la información a y dentro de un equipo de desarrollo es mediante la conversación cara a cara.

7. El software funcionando es la medida primaria de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinantes, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de manera indefinida.

9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.

10. La simplicidad, el arte de maximizar la cantidad de trabajo que no se hace, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de los equipos autoorganizativos.

12. En intervalos regulares, el equipo reflexiona acerca de cómo volverse más efectivos, y luego entona y ajusta en consecuencia su comportamiento.

Para estudiar el proceso de implementación en el marco de los métodos ágiles, se estudiarán las estructuras conceptuales de dos de ellos. El primero es el representante más conocido de los métodos ágiles, Programación Extrema (*eXtreme Programming*, XP por sus siglas en inglés) (Beck, 2000). Finalmente, se revisará el método AgileUP (Ambler, 2001) como una instanciación ágil del Proceso Unificado.

4.1 Programación Extrema (XP)

Programación Extrema (XP, por *eXtreme Programming*) es una disciplina de desarrollo de software basado en los valores de simplicidad, comunicación, retroalimentación y valor (coraje), que propone a los equipos de trabajo la implantación de prácticas simples que les permite recibir retroalimentación de la situación actual del proyecto y ajustar las prácticas a dicha situación específica (Jeffries, 2001).

La Programación Extrema fue creada por Kent Beck durante su trabajo en el proyecto de Nómina del Sistema de Compensación Comprehensiva de Chrysler (C3) (Wells, 2006). La principal meta de XP es reducir el costo del cambio (Beck, 2000), mientras que los métodos tradicionales buscan que los requisitos sean determinados y establecidos al comienzo del proyecto de desarrollo, y procurando que así permanezcan de ahí en adelante. XP propone, en consonancia con otros métodos ágiles, que los cambios continuos sobre los requisitos son un aspecto natural, ineludible y deseable de los proyectos de desarrollo de software. Así pues, partiendo de que la adaptabilidad de un proyecto a requisitos cambiantes es un enfoque más realista que intentar definirlos todos e invertir un gran esfuerzo en controlar sus cambios. XP busca la reducción del costo del cambio por medio de la aplicación de sus valores, principios y prácticas.

4.1.1 Estructura conceptual de la Programación Extrema

La Programación Extrema se conceptualiza primordialmente con la definición de sus valores y sus prácticas principales. Es esta definición simple junto al curso básico de las acciones llevadas a cabo lo que le da el puesto como metodología ágil por excelencia.

Entre los valores que este método promueve se tienen: 1) *Comunicación*, promoviendo que esta forme parte del comportamiento del día a día en un equipo de trabajo, tanto en la frecuencia con que ocurre como con la calidad de las comunicaciones. 2) *Simplicidad*, consiste en la inclusión de las soluciones más simples que puedan funcionar, haciendo que el sistema sea más fácil de entender y ser comunicado entre el equipo y con el cliente. 3) *Retroalimentación*, que es la principal estrategia de mitigación de riesgos que usa este método, en escalas tan diversas como la retroalimentación rápida que las pruebas unitarias dan al programador, como la retroalimentación, que también se procura hacerla tan rápida como sea posible, al desplegar una nueva versión de la aplicación en producción al cabo de unas pocas semanas. 4) *Coraje*, entendido como valor u osadía, propone la importancia de atreverse a hacer cambios y tomar decisiones duras cuando se sabe que es lo correcto aunque una estrategia más conservadora optaría por negociar una salida menos riesgosa. 5) *Respeto* en el entorno del equipo, entre los programadores y con el cliente, tanto a nivel de los individuos como asociado al trabajo que cada uno realiza.

Por su parte, las prácticas son las siguientes (Beck, 2000):

- *Juego de Planeación*: Es el proceso principal de planeación en XP. Consiste en una reunión realizada a períodos cortos y regulares, donde el cliente y el equipo de desarrollo determinan de manera rápida el alcance de una iteración con base en el valor de negocio y la estimación del esfuerzo de los requisitos.
- *Pequeños despliegues*: Colocar un sistema simple en producción de manera rápida, y desplegar nuevas versiones haciendo uso de iteraciones cortas.
- *Metáfora del sistema*: Es un concepto de nomenclatura de clases y métodos que deberían hacer fácil a un equipo de trabajo el determinar la funcionalidad de una clase o método particular. Se propone guiar todo el desarrollo haciendo uso de una historia simple compartida acerca de cómo el sistema completo funciona.
- *Diseño simple*: El sistema debe ser diseñado tan simple como sea posible en cualquier momento, partiendo de la premisa de implementar la solución más simple que pueda funcionar. Cualquier complejidad adicional debe removerse en cuanto sea detectada.
- *Programación en pares*: Esta práctica promueve que dos desarrolladores realicen un esfuerzo de desarrollo en una sola estación de trabajo.
- *Desarrollo orientado a las pruebas*: Consiste en el diseño e implementación del código de pruebas previo a la creación del componente a ser probado. De esta manera, los casos de pruebas pueden ser usados como mecanismo de diseño del código del producto. Adicionalmente, los clientes escriben casos de pruebas que serán usados como especificación de los requisitos y criterio de aceptación de los mismos.
- *Refactorización del Diseño*: Los programadores deben reestructurar el diseño interno del sistema sin cambiar su comportamiento con el fin de remover duplicidad, mejorar la legibilidad, la simplicidad o agregar flexibilidad en el código fuente.
- *Programación por Pares*: Todo el código de producción es escrito por dos programadores trabajando en un solo computador.
- Pertenencia colectiva del código: Significa que todos los desarrolladores son responsables por todo el código, lo que a su vez significa que todos tienen el derecho de modificar cualquier parte del código. XP considera que el código pertenece al proyecto, no a un programador en particular.
- *Integración continua*: Integrar y generar el sistema completo varias veces al día, cada vez que una tarea se haya completado.
- *Ritmo sostenible*: El concepto es que los desarrolladores no deberían trabajar más de 40 horas semanales, así como el que si una semana genera sobretiempo, para la próxima semana dicho desarrollador no debería incluir más sobretiempo.
- *Cliente como parte del equipo*: Consiste en la inclusión de un cliente o usuario como parte del equipo de trabajo, disponible en tiempo completo para responder cualquier pregunta que se tenga.
- *Estándares de codificación*: Deben ser acordados a partir de un conjunto de reglas que el equipo completo de desarrollo debe estar de acuerdo en acatar a lo largo de todo el proyecto. Los programadores deben escribir todo el código de acuerdo a estos estándares, poniendo énfasis en la comunicación a través del código.

4.1.2 Proceso de Implementación en el contexto de la Programación Extrema

El Proceso de Implementación está englobado en lo que Beck (Beck, 2000) describe como un episodio de desarrollo protagonizado por equipos de dos programadores. En dicho episodio se observa la autoorganización de los equipos, la asignación de las tareas a realizar, la comunicación descentralizada con el cliente, la especificación funcional en forma de casos de prueba, el diseño evolutivo en forma de refactorización, el diseño detallado en forma de pruebas unitarias y el proceso de integración de los componentes como una actividad más del día a día en lugar de un proceso separado y ajeno a la implementación de los mismos componentes.

XP promueve la ocurrencia reiterativa de estos episodios en las prácticas que propone para la construcción del software (Beck, 2000, Wells, 2006). De ellas, las siguientes prácticas están inherentemente asociadas al proceso de construcción como lo son:

- Programación orientada a pruebas (TDD)
- Mejoras evolutivas del diseño o Refactorización
- Integración Continua
- Pertenencia colectiva del código
- Diseño simple
- Adhesión a estándares de codificación

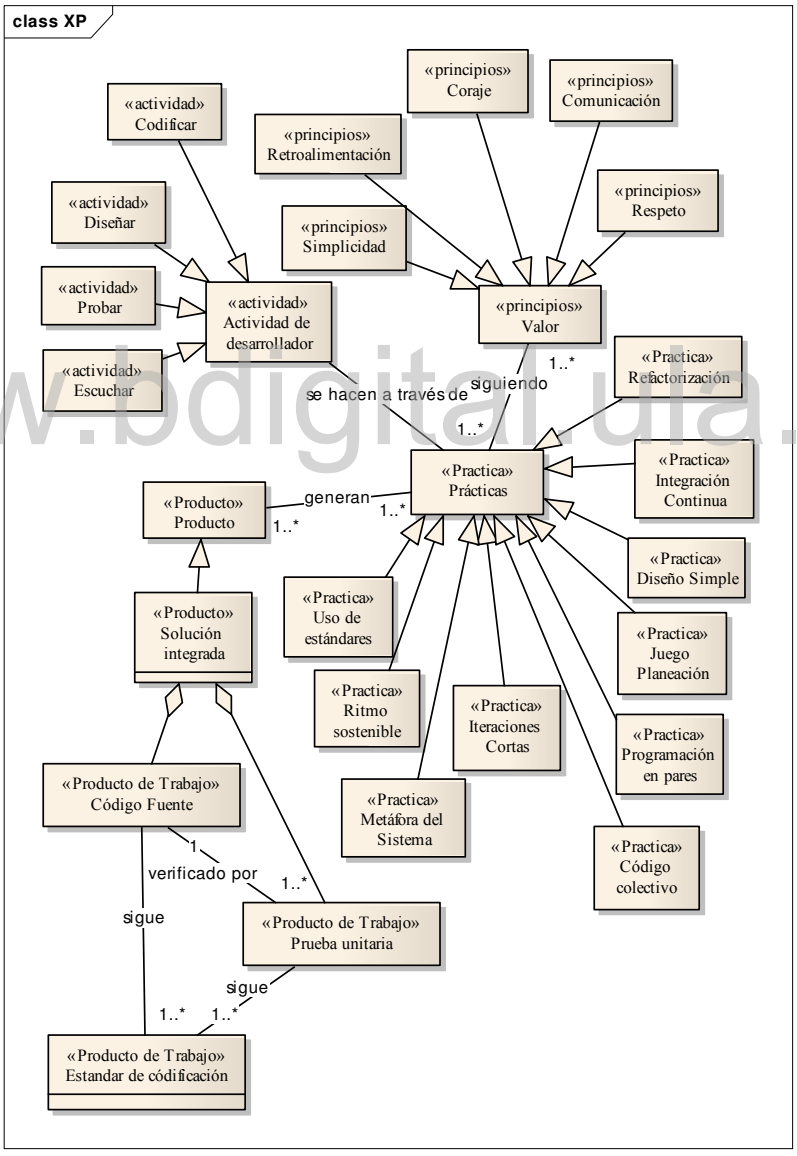


Figura 7. Modelo Conceptual del Proceso de Implementación de Software según Programación Extrema (XP)

Los procesos de Verificación están asociados con la revisión constante del producto, gracias a la práctica de Programación en Pares, y la ejecución continua de pruebas unitarias. El proceso de Integración se refleja en la práctica de Integración Continua, promoviendo que siempre una última versión del código a la cual se integran los componentes ya probados de modo unitario. En la figura 7 se muestran los conceptos manejados por la disciplina XP.

4.2 AgileUP

El Proceso Ágil Unificado (*Agile UP*, por sus siglas en inglés) es un enfoque de desarrollo de software basado en el *Unified Process* (UP) (Ambler, 2005, Larman, 2001), y procura ser una versión simplificada del *Rational Unified Process* (RUP). Este método describe un enfoque simple para desarrollar aplicaciones de negocio usando técnicas y conceptos ágiles a la vez que se mantiene compatible con RUP. Ambler, quien tiene amplia experiencia con RUP gracias a su trabajo extendiendo este método a través del *Enterprise Unified Process™* (EUP), comenzó a trabajar en el proceso de agilizar a RUP incluyendo prácticas ágiles de modelado, las cuales están recopiladas en su libro Modelado Ágil (Ambler, 2002). Allí propone un método de modelado de software que se encuentra estructurado de manera similar a la Programación Extrema, haciendo uso de valores y prácticas. La experiencia lograda con RUP y Modelado Ágil culminó por la presentación de AgileUP en el año 2005.

4.2.1 Estructura conceptual de AgileUP

AgileUP sigue una estructura basada en similar a RUP, siguiendo una organización de *Fases* secuenciales en el tiempo (Definición, Elaboración, Construcción y Transición) que contienen numerosas iteraciones y donde las actividades se encuentran agrupadas lógicamente en *Disciplinas*, que se asemejan a los *Workflows* propios de RUP. El ciclo de desarrollo del *Agile UP* es serializado en los procesos generales, iterativo a nivel detallado y entrega versiones incrementales del producto a lo largo del tiempo (Ambler, 2005).

Las *Disciplinas* son desarrolladas de manera iterativa, definiendo las actividades que los miembros del equipo de desarrollo deben realizar para construir, validar y entregar un software funcional que satisfaga las necesidades de sus clientes o interesados. Cada disciplina especifica un flujo de trabajo (*Workflow*), una serie de recomendaciones y las actividades primarias por cada fase. Entre las disciplinas que se pueden encontrar en AgileUP se tienen: Modelado, Implementación, Pruebas, Despliegue, Gestión de la Configuración, Gestión del Proyecto y Entorno.

De manera congruente con la tónica propia del desarrollo ágil, AgileUP presenta sus propios principios, entre los que se listan los siguientes:

- El equipo sabe lo que está haciendo.
- Simplicidad
- Conformidad con los principios del Manifiesto Ágil.
- Foco en actividades de alto valor
- Independencia de las herramientas, seleccionando las herramientas que mejor se adapten al trabajo en manos.

- Adaptar el método para cubrir las necesidades en manos.

AgileUP maneja los conceptos de entregables, productos de trabajo empresariales y otros productos de trabajo. Un producto de trabajo consiste en cualquier elemento de valor que sea producido, usado, modificado o consumido durante el desempeño de un proceso. Los entregables son aquellos productos de trabajo que deben ser producidos permanentemente. Los otros productos de trabajo son aquellos que posiblemente sean descartados porque no se requiere mantenerlos en el tiempo; y los productos de trabajo empresariales son aquellos que son mantenidos dentro de la organización TI y es compartido entre proyectos. Las prácticas recomendadas para mantenerse ágil respecto a la generación de los diversos productos de trabajo son:

- Mantener los productos de trabajos tan simples y concisos como sea posible.

- Se requiere mucha menos documentación que la que se puede pensar.

- Trabaja de manera cercana a la gente para quien se está creando un producto de trabajo de manera que se produzca sólo aquello que realmente se necesita.

- Los documentos ágiles son apenas suficientemente buenos para la tarea que se realiza.

- Producir un documento es la peor forma de comunicar información; varias personas discutiendo alrededor de un pizarrón es la mejor.

- Usar herramientas simples tales como pizarrones, papel y wikis para modelar y capturar documentación.

- Considerar el usar las plantillas libres o de fuente abierta como una base para crear las propias.

4.2.2 Proceso de Implementación en el contexto de AgileUP

Al igual que UP, una de las fases contempladas en el ciclo de vida del producto es la *Fase de Construcción*. En esta fase se desarrolla el sistema de software hasta que se encuentra listo para las pruebas de pre-producción. En las fases previas, la mayor parte de los requisitos han sido identificados y la arquitectura del sistema se encuentra en la línea base. Así que, el énfasis se concentra en priorizar y en entender los requisitos, en hacer tormentas de ideas sobre las soluciones propuestas y en codificar y probar el producto. En caso de ser necesario, las primeras versiones del producto son desplegadas, interna o externamente, con el objeto de obtener retroalimentación por parte del usuario. La *Fase de Construcción* culmina cuando el equipo de desarrollo alcanza a desarrollar un producto operacional.

El Proceso de Implementación de software se encuentra englobado dentro de la *Disciplina de Implementación*. Esta disciplina tiene la meta de transformar los modelos en código ejecutable y realizar un nivel básico de pruebas, refiriéndose con esto a las pruebas unitarias. El flujo de trabajo de esta disciplina en los siguientes subprocesos y tareas:

1. Desarrollo de la aplicación: se realiza la codificación y prueba del producto de software final. Es llevado a cabo por el desarrollador, usando información aportada por los interesados y por el modelador ágil. Entre las tareas que se llevan a cabo en este proceso se encuentran: Escribir las pruebas unitarias; Escribir el código de producción; Ejecutar las pruebas unitarias; Perfilar el desempeño del sistema y Gestionar las dependencias en el código

2. Construir el sistema: consiste en la compilación y el empaquetado del sistema para su despliegue. Las tareas contempladas son: Compilación (Integración) continua del sistema; Promover al entorno de integración del proyecto.

3. Desarrollar la Base de Datos: realizada por el Administrador de Base de Datos. Sus tareas son: Evolucionar el esquema de base de datos y Gestionar los derechos de acceso.

Entre las actividades primarias de la *Disciplina de Implementación* durante la *Fase de Construcción* se tienen: 1) Desarrollar evolutivamente la lógica del dominio, 2) Desarrollar evolutivamente la interfaz de usuario, 3) Desarrollar evolutivamente el esquema de datos, 4) Desarrollar interfaces a sistemas legados; y 5) Escribir *scripts* de conversión de datos.

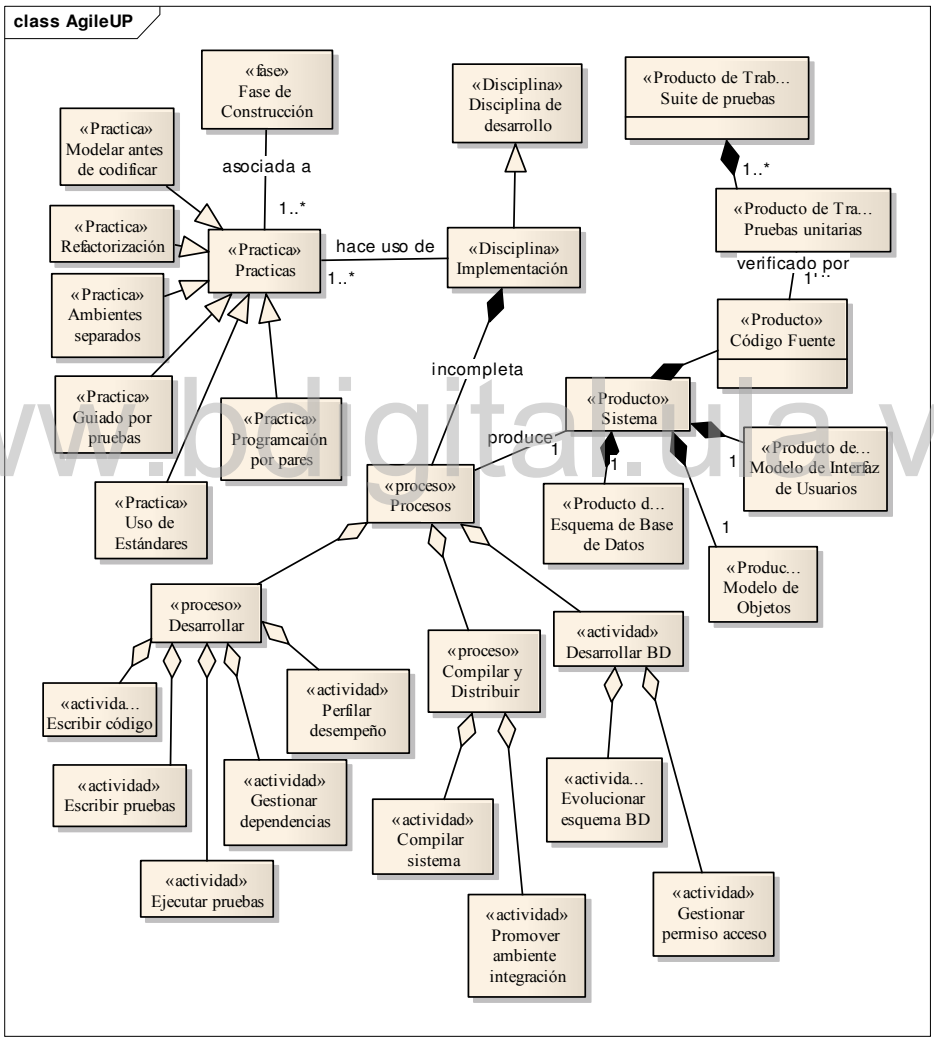


Figura 8. Modelo Conceptual del Proceso de Implementación de Software según AgileUP

Entre tanto, las recomendaciones para la *Disciplina de Implementación* se tiene el incluir las siguientes prácticas: Programación por pares, Desarrollo guiado por pruebas (*Test Driven Development*, TDD por sus siglas en inglés), Integración Continua, Modelar antes de codificar, Adoptar y seguir un estándar de codificación,

Refactorízación del código y de los esquemas de bases de datos y tener ambientes separados para el desarrollo, las pruebas y la producción.

En AgileUP, el proceso de integración de componentes se traduce en la actividad continua de compilación del sistema. Esta compilación se debe realizar cada vez que el código fuente cambia. Así, el repositorio de código fuente se encuentra, en todo momento, en un estado comparativamente estable respecto a las nuevas funcionalidades que han sido agregadas al producto. El conjunto de conceptos de *AgileUP* se muestra en la figura 8.

5 Proceso de implementación orientado al plan vs. Proceso de implementación ágil

Luego de haber analizado la conceptualización del Proceso de Implementación en cada uno de los métodos y modelos representantes de los enfoques ágil y orientado al plan, se realiza una comparación detallada de los modelos conceptuales extraídos. Con esto se quiere establecer las similitudes en los conceptos, prácticas y actividades recomendadas o prescritas en ambas perspectivas, así como aquellos elementos que son propios a cada una de ellas y que las distinguen entre sí.

La tabla 1 resume los resultados del análisis realizado sobre la base de los procesos de: diseño detallado, codificación (programación), verificación (pruebas) e integración.

Tabla 1. Resumen comparativo de las perspectivas

Fase	Similitudes	Diferencias
Diseño detallado	• Necesidad de diseño detallado previo a la implementación de un componente	• Documento de Diseño detallado (orientado al plan) vs. Creación de la prueba unitaria y código legible y documentado como el diseño detallado (ágil). • Productos de trabajo actualizados (orientado al plan) vs. Temporalidad de los productos de trabajo (ágil).
Programación	• Desarrollo de pruebas unitarias y la práctica de desarrollo guiado por pruebas • Existencia y adhesión a estándares de codificación.	• Documentación exhaustiva (orientado al plan) vs. Documentar sólo lo necesario (ágil) • Refactorización – a discreción - del código como método evolutivo de incremento de la calidad técnica (ágil).
Verificación	• Revisión regular del código fuente.	• Revisión estructurada de código (orientado al plan) vs. Revisión constante de pares (ágil).
Integración	• Comprobación del producto integrado a través de pruebas de integración.	• Proceso separado de Integración explícita de Componentes (orientado al plan) vs. Integración Continua (ágil).

5.1 Similitudes

- Ambas perspectivas concuerdan en establecer como requisito previo, la existencia de algún tipo de diseño detallado antes de iniciar la actividad de codificación del producto de software. No se considera si este diseño tiene asociado una especificación técnica formal o informal.

- Consideran como elemento crucial la inclusión de pruebas unitarias sobre el producto y la definición del modelo de procesos de desarrollo de software basado en las pruebas, ya sea de manera explícita (ágiles) o de manera implícita (diseño de los casos de prueba antes de codificar en los tradicionales).

- Una y otra perspectiva promueven y proponen la existencia de estándares de codificación y es vital la adhesión a dichos estándares por parte de los desarrolladores.

- El código fuente es objeto de revisiones regulares para asegurar que es correcto, su calidad y el grado de adhesión a los estándares de codificación predefinidos.

- La correcta ejecución del proceso de integración se comprueba haciendo uso de pruebas de integración que se aplican al producto resultante.

5.2 Diferencias

- En los métodos orientados al plan, el diseño detallado es necesariamente un documento o un conjunto de ellos, y hasta un paquete que incluye documentos y varios modelos. En los métodos ágiles, el diseño detallado se logra con la creación de la prueba unitaria y con el código fuente, legible y documentado, sin que se requiera algún documento o artefacto adicional.

- En los representantes tradicionales el proceso de integración se realiza *a posteriori* a la codificación de los componentes de la aplicación, en tanto que en los representantes ágiles este proceso es sustituido por la práctica de integración continua.

- La revisión técnica del código es una práctica propuesta por el modelo disciplinado *CMMI* para realizarle entre pares, en el Método *WATCH* se realiza por un grupo de expertos organizados por el grupo de V&V. En el caso de los representantes ágiles, la revisión de pares se plantea de manera continua (programación por pares) y regular para asegurar la calidad del código, en la forma de programación por pares, sin que por ello se requiera una revisión adicional externa.

- Los representantes disciplinados presentan una diversidad de productos de trabajo, entre los cuales se encuentran los documentos y modelos de diseño, los cuales son resguardados y deben ser actualizados constantemente ante cualquier cambio que ocurra. En los representantes ágiles estudiados, existe la temporalidad de los productos de trabajo, en algunos casos se prefiere descartarlos cuando ya no se necesitan y en algunos casos simplemente no se prescriben como necesarios, reduciendo la carga adicional de resguardo y actualización ante la inminencia de cambios sobre el producto.

- Sin excepción, todos los modelos disciplinados mencionan el proceso de documentación como parte de la implementación. Esta documentación es operativa e incluye manuales de sistema, de usuario, ayuda en línea y de diseño. En el caso particular de *AgileUP* se contempla solo la documentación que debe ser considerada entregable (al cliente) más no los productos de trabajo internos; el método XP no propone documentación alguna.

Los métodos ágiles incluyen la refactorización del código, es decir, el cambio a discreción del desarrollador para hacer que un código complejo realice la misma funcionalidad que hace actualmente de manera más sencilla, para el incremento evolutivo de la calidad del código. Esta actividad discrecional del desarrollador no aparece mencionada o propuesta por ninguno de los representantes disciplinados.

Esta última diferencia abre un poco más la brecha entre ambas perspectivas, resaltando a la faceta humana de quienes desarrollan el producto como elemento diferencial clave. Es decir, la actividad de refactorización que promueve el incremento de la calidad del código como una iniciativa a discreción del programador. Si bien ésta no tiene prohibición expresa en ninguno de los representantes disciplinados, su aplicación se puede ver obstaculizada por el cambio cultural que representa para una organización tradicional (quienes aplican habitualmente el enfoque disciplinado), el hecho de otorgar tal poder de acción a un desarrollador. Dado que este documento se concentra en la comparación a nivel de conceptos asociados a los procesos, actividades, prácticas y productos manejados, el estudio del aspecto recurso humano en el desarrollo de software sale fuera del ámbito de competencia de este artículo.

5.3 Modelo conceptual de un proceso de implementación de software ágil y disciplinado

Basado en las disposiciones definidas tanto en el Manifiesto Ágil como en los modelos que preconizan las mejores prácticas asociadas a la implementación de software de modo disciplinado, se define un modelo que contempla los conceptos que caracterizan a un proceso de implementación ágil y disciplinado. En la tabla 2 se resume los subprocesos, las actividades y las prácticas que se consideran para definirlo. Es importante aclarar que, en la mencionada tabla, no hay ningún modelo de ejecución o secuencia implícita o explícita para estos subprocesos.

Tabla 2. Tabla de subprocesos actividades y prácticas en el modelo integrado de implementación

Subprocesos	Actividades	Prácticas
Construcción de Software	Diseño detallado de componentes Codificación Documentación de código Documentación de soporte del software	Uso de sistemas de control de versiones de código Definición y adopción de estándares de codificación y construcción en general Refactorización del código Asegurar legibilidad del código mediante nomenclatura y comentarios
Verificación y Validación de Software	Pruebas Realización de revisión de pares (Peer Review) Realización de inspecciones técnicas	Registro de incidencias del revisión de pares en base de datos de incidencias Desarrollo guiado por pruebas. Uso de pruebas unitarias y de integración automatizadas

Integración de Software	Ensamblar componentes de producto Realizar pruebas de integración	Integración continua Uso de guiones o *scripts* de automatización de versiones ejecutables

La figura 9 muestra que el Proceso de Implementación propuesto tiene como subprocesos los de Construcción, Integración y V&V. Las actividades definidas para cada proceso son llevadas a cabo por los miembros del equipo de desarrollo.

El subproceso de Construcción tiene como actividades el diseño detallado de los componentes a implementar, la codificación y la documentación - ya sea en el código fuente, la generación de documentación de operación y alguna otra documentación que haya sido establecida como entregable del proyecto.

El subproceso de Integración abarca las actividades de planificación de la integración de la aplicación, el ensamblado propiamente dicho de los componentes a ser integrados y la ejecución de las pruebas de integración. Estas actividades son realizadas en el contexto de la práctica de *Integración Continua*. Las pruebas de Integración se realizan como parte de las pruebas unitarias. Estas pruebas unitarias están asociadas a la práctica propuesta tanto por métodos disciplinados y ágiles, es decir, en el desarrollo de software basado en las pruebas del producto.

Finalmente, el subproceso de V&V maneja las actividades de realización de revisión de pares (*Peer Review*) y la inspección técnica que debe ser realizada por miembros externos al equipo de desarrollo.

El proceso global de implementación se apoya en estándares los cuales se pueden conseguir a través de medios de públicos (internet) o privados (experiencia interna de la organización), el estilo de codificación y la organización de los componentes. En relación a las herramientas que se recomienda usar para apoyar la realización de los subprocesos contemplan el repositorio de código o sistema de control de versiones, pasando por la base de datos de incidencias y los guiones de generación del despliegue y etiquetado, hasta llegar al servidor de integración continua, asociado a la práctica del mismo nombre. Además de esta práctica, se tiene el Desarrollo guiado por pruebas y la refactorización del código, que consiste en el cambio de implementación preferiblemente sin afectar la interfaz ni el funcionamiento del componente para obtener un código más limpio, legible y de mayor calidad.

5.4 Lineamientos en los que se basa la propuesta de un proceso de implementación ágil y disciplinado

Considerando que, además de las diferencias conceptuales observadas entre los modelos y los términos determinados para cada perspectiva, y de las diferencias pragmáticas al momento de usar modelos pertenecientes a alguna de ellas, se debe tener en cuenta la influencia que la cultura organizacional de quienes aplican los diferentes métodos o modelos tiene sobre el proceso de implementación; es decir, la experiencia y el modo de trabajo de los miembros que constituyen el equipo de desarrollo (Fowler y Highsmith, 2001) influye en la factibilidad de realizar o no una práctica de software. Así, se opta por basar la propuesta de un proceso de implementación ágil y disciplinado en la inclusión de los principios y prácticas propuestas por los modelos ágiles dentro de los procesos, actividades y prácticas de los modelos orientados al plan. En relación a la documentación, se decide minimizarla a nivel interno manteniendo los documentos entregables y los de relación con el cliente. Estos lineamientos se resumen a continuación:

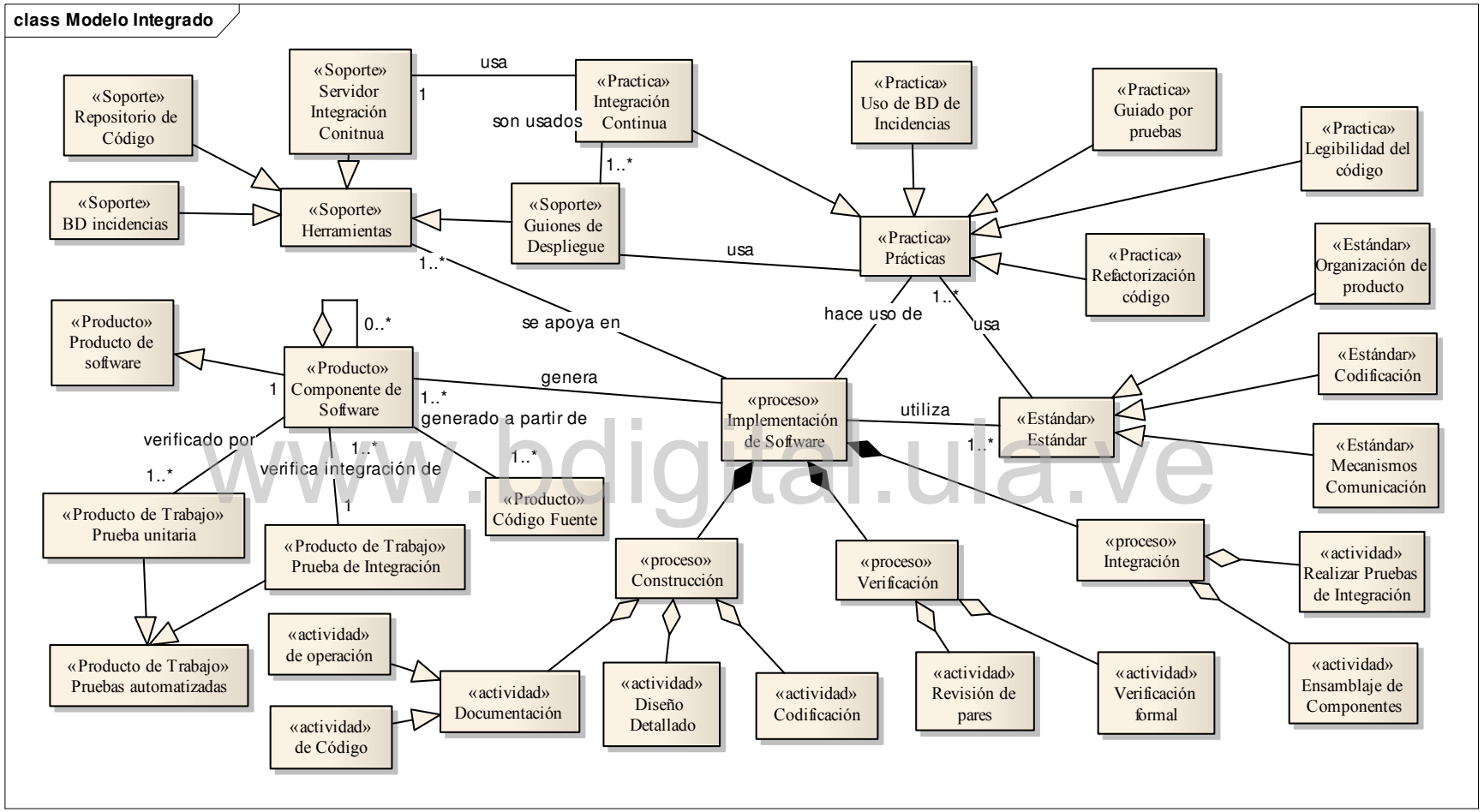


Figura 9. Modelo Conceptual Integrado de la Implementación de Software ágil y disciplinada

- Establecer un mínimo de documentación básica o imprescindible, dejando el resto como documentación opcional. La decisión de generar algún documento opcional va dada por algún requisito entregable al cliente, o por el valor que se considere éste agrega al componente de software en desarrollo. Debe tenerse presente que todo producto de trabajo que se mantenga agrega esfuerzo y carga de trabajo durante el desarrollo lo que pudiera ir en detrimento de la agilidad del proceso de implementación, y también, en fases posteriores de mantenimiento y actualización del producto ya operativo.

- En concordancia con la mejora anterior, se pretende *minimizar* la generación de especificaciones y de *modelos del diseño detallado*, especialmente, si estos deben ser actualizados por separado al código fuente y a las pruebas unitarias. Los métodos ágiles han resuelto este problema definiendo el código fuente y las pruebas unitarias como parte del diseño detallado, sin embargo, se sugiere apoyar el proceso a través de una herramienta de generación de código que tome como entrada el modelo o la especificación de diseño detallado.

- Asumir que el proceso de Integración de Componentes se hace como una práctica de *Integración Continua*, realizada regularmente con cada nuevo incremento por cada desarrollador. Las pruebas de integración se convierten en pruebas unitarias automatizadas realizadas por el desarrollador. Esto requiere tener la capacidad de generar versiones y ejecutar las pruebas unitarias de manera automatizada, con algún *script* de generación de versiones y despliegue.

- Incluir y promover la práctica de *Refactorización* del código. Esto es un paso importante para la evolución de la calidad del código fuente, y promueve que el desarrollador, no sólo esté pensando en la solución de la funcionalidad a completar o error actual a corregir, sino también en constante inspección de su trabajo anterior, lo cual puede traducirse en eliminación de errores incluso antes de ser detectados.

- Aunque puede parecer un elemento obvio, es importante que se enfatice en la importancia de la *calidad y la legibilidad del código fuente*. Un código con estas características puede ser usado como su propia documentación y modelo, además de facilitar el mantenimiento y la inclusión de nuevos desarrolladores a mitad del proyecto, lo cual es una realidad que debe aceptarse.

6 Conclusiones de este capítulo

Este trabajo de investigación define dentro del Proceso de Implementación de Software las actividades asociadas al diseño detallado, aprovisionamiento, codificación, documentación, verificación y validación e integración de componentes.

La agilidad y la disciplina no son atributos incompatibles, ni solamente asociados a un enfoque específico de desarrollo de software. Los modelos de desarrollo orientados al plan procuran una definición bien elaborada de los procesos, sea en un nivel genérico o detallado, y generan sinergia en el uso conjunto de modelos de mejora de procesos, mejores prácticas asociadas a la gestión de proyectos y uso extensivo de los estándares definidos para modelos de desarrollo de software.

Por su parte, los métodos ágiles tienen como patrón común el Manifiesto Ágil (Fowler y Highsmith, 2001), que define un conjunto de valores y principios asociados a la flexibilidad, adaptabilidad al cambio, orientación a la gente y la autoorganización de los equipos. Estos métodos promueven la generación de características emergentes a partir de la implementación de prácticas básicas, y promueven la poca ceremonia y poca definición de procesos en sus implementaciones.

Existen similitudes y diferencias entre estos dos enfoques, sin embargo esto no afecta la posibilidad de definir un modelo conceptual de Proceso de Implementación de Software que pueda ser exhibir los atributos de agilidad y disciplina.

CAPITULO III: Ingeniería de Métodos Situacionales

En este capítulo se presentan los antecedentes teóricos de la Ingeniería de Métodos Situacionales, cuyo enfoque provee el marco de referencia para el modelado de los fragmentos de productos y procesos ágiles y disciplinados propuestos en este trabajo. De igual manera se define la estrategia de representación para el modelo de productos y procesos que será usado en los capítulos IV y V, respectivamente.

En la primera parte se presenta un resumen de los conceptos básicos asociados a la Ingeniería de Métodos Situacionales, sus principios y motivación. La sección dos contempla los meta-modelos de métodos usados en la construcción de métodos situacionales, mientras que en la sección tres se expone un resumen de los enfoques y técnicas usadas para construir tales métodos situacionales. En la sección cuatro, se presenta la estrategia planteada para la creación de un método situacional que se ajuste al ámbito que se ha definido previamente para el proceso de implementación de software y que contenga los elementos de agilidad y disciplina que se describieron en el capítulo II, sección 5.

1 Bases de la Ingeniería de Métodos Situacionales

1.1 Conceptos Básicos

La *Ingeniería de Métodos* se define como la disciplina ingenieril cuyo objeto es el diseñar, construir y adaptar métodos, técnicas y herramientas para el desarrollo de sistemas de información (Brinkkemper, 1996). El objeto de estudio principal de la Ingeniería de Métodos, como su nombre lo indica, es el *método*, el cual es conceptualizado como un enfoque de ejecución de un proyecto de desarrollo que consiste en directivas y reglas estructuradas de manera sistemática en actividades de desarrollo que se corresponden con productos de desarrollo (Brinkkemper, 1996). Un método aporta los conceptos para describir el producto y las reglas de conducta metodológica para desarrollar un producto de software de calidad con una eficiencia razonable (Rolland, 2005).

La *Ingeniería de métodos situacionales* es el sub-área de la ingeniería de métodos que se dedica a la construcción controlada, formal y asistida por computadora de métodos situacionales a partir de fragmentos (Harmsen, 1997). Uno de los primeros temas de investigación que se tuvo en el seno de la Ingeniería de Métodos está asociado a los *métodos situacionales*, que puede definirse como un método de desarrollo de sistemas de información que es creado y ajustado para satisfacer las condiciones dadas por la situación del proyecto (Brinkkemper, 1996, Harmsen, 1997, Arni-Bloch, 2005).

Distintos autores especialistas en la materia coinciden en que la capacidad de un método de satisfacer las necesidades de un proyecto en una situación dada requiere la modularización de los métodos en forma de piezas reusables de método y guías para el uso y ensamblaje de los mismos en métodos, luego de su búsqueda y recuperación desde un repositorio de fragmentos llamado base de métodos. Estas piezas reusables de método pueden ser *fragmentos de método* o *chunks de método* (Mirbel y Ralyté, 2006).

Un *fragmento de método* es una pieza coherente de un método de desarrollo de sistemas de información (Brinkkemper, 1996). Harmsen lo define como una descripción de un método de ingeniería de sistemas de información, o cualquier parte coherente de este (Harmsen, 1997). Los fragmentos de método pueden ser de dos tipos: Fragmentos de proceso y fragmentos de producto. Los fragmentos de producto modelan la estructura de los productos, entre los que se encuentran los entregables, diagramas, tablas y modelos, de un método de desarrollo de sistemas, mientras que los fragmentos de proceso modelan al proceso de desarrollo (Brinkkemper, 1996).

Por su parte, un *chunk de método* es una parte cohesionada, autónoma e interoperable de un método (Arni-Bloch, 2005). Un *chunk de método* está conformado por uno o más fragmentos de producto y uno o más fragmentos de proceso.

1.2 Motivación de la Ingeniería de Métodos Situacionales

De acuerdo con (Rolland, 2006), un método que ha sido probado exitosamente en un dominio específico no siempre puede ser aplicado sin modificación a otro dominio. Rara vez los métodos de desarrollo son aplicados al pie de la letra y los desarrolladores de aplicaciones se ven en la necesidad adaptar los métodos a los contextos de los proyectos en que participan. Sin embargo, las prácticas usadas en los métodos adaptados suelen venir de otros métodos, ya que no tiene sentido inventar una manera nueva de resolver un problema específico si ya alguien en el pasado lo resolvió exitosamente. Se ha pretendido que métodos de carácter universal resuelvan este tipo de situaciones, pero este tipo de métodos que se proponen como 'a prueba de balas' terminan penalizando a muchos proyectos, proponiendo llevar a cabo actividades y procesos que pueden no ser necesarios en todos los casos. Rolland asegura que la ingeniería de métodos situacionales se ve justificada por la sistematización del desarrollo de sistemas de información cada vez más complejos y que deben ser realizados a menor costo, requiriendo así el uso de métodos de desarrollo, y la incapacidad de los métodos existentes de responder a las necesidades diversas y constantemente cambiantes en el desarrollo de software. Esto justifica que se propongan nuevas maneras de definir, construir y adaptar los métodos de desarrollo.

1.3 Principios de la Ingeniería de Métodos Situacionales

Rolland (Rolland, 2006) propone cuatro principios de la ingeniería de métodos situacionales:

1. **Principio de meta-modelado**: Este principio rige la descripción de los métodos. Establece que los modelos que conforman un método deben ser a su vez meta-modelados. Un modelo de método está conformado por un meta-modelo de productos (que permite describir modelos de productos) y un meta-modelo de procesos (usado para describir un modelo de procesos).

2. **Principio de reutilización**: Plantea la conformación de métodos a partir de componentes de métodos existentes, ya sean partes de modelos de producto o partes de modelos de procesos. Similar a la reutilización de componentes de software, busca aumentar la productividad en la creación y adaptación de métodos a un contexto dado y aumentar la calidad del método creado gracias al uso de prácticas cuya efectividad ha sido comprobada en proyectos previos.

3. **Principio de modularidad**: Gracias a este principio, un método es visto como un conjunto de componentes de métodos reutilizables, que pueden ser definidos de manera detallada y que deben presentar ciertos atributos de calidad como cohesión, autonomía e interoperabilidad. Un método cuyos componentes han sido definidos respetando este principio, presenta la posibilidad de intercambiar un componente diseñado para cumplir un objetivo dado siguiendo una estrategia A, por otro componente que tiene el mismo objetivo pero haciendo uso de una estrategia A'.

4. **Principio de flexibilidad**: Este principio se encuentra en el corazón de la ingeniería de métodos donde la problemática central es la adaptación de métodos de acuerdo a los factores contextuales o situación específica de un proyecto. La flexibilidad de un método radica en su capacidad inherente para ser adaptado a las condiciones que un proyecto presenta.

1.4 Espectro de los Métodos Situacionales

Una manera de mostrar la evolución hacia los métodos situacionales la define Harmsen en (Harmsen, 1997) como el espectro de los métodos situacionales, donde se observan las diferentes propuestas para responder a la necesidad de usar métodos de desarrollo distintos de acuerdo a cada situación. Este espectro incluye distintos enfoques que van desde el extremo de los métodos rígidos que no permiten adaptación hasta el uso completo de los conceptos de la Ingeniería de Métodos Situacionales:

1. Uso de métodos rígidos

Un método rígido es un método estándar de desarrollo de software que en su definición no deja espacio para la adaptación a situaciones distintas. Sus estándares, técnicas, procedimientos de modelado y filosofía están completamente predefinidos y no incluye guías para la adaptación del método.

2. Selección a partir de métodos rígidos

Consiste en la selección entre varios métodos rígidos basado en la situación general del proyecto. Luego de seleccionar el método considerado más apropiado a la situación, es usado sin modificación alguna.

3. Selección de rutas predefinidas dentro de un método específico

Este enfoque requiere de un método que permita la selección de diferentes rutas de ejecución en su definición. Tales rutas pueden ser fijas y predefinidas o dinámicas, permitiendo con esto que nuevas rutas puedan ser agregadas en el tiempo.

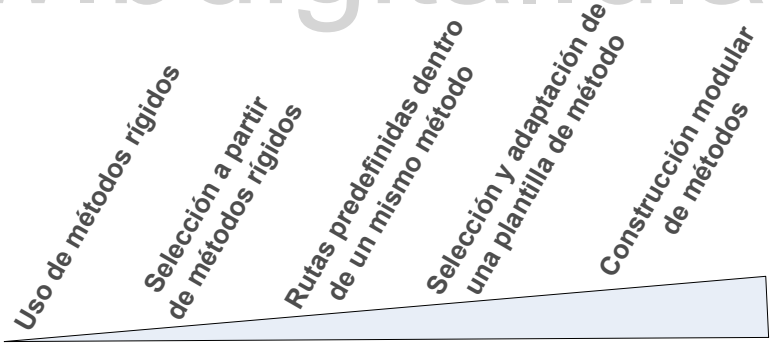


Figura 10. Espectro de los Métodos Situacionales

4. Selección y adaptación de una plantilla de método

Este enfoque busca una solución más dinámica y adaptable, que permite no sólo seleccionar entre diferentes enfoques sino también adaptar y refinar el enfoque seleccionado de acuerdo a la situación del proyecto entre manos. Aquí se involucra la selección de un modelo global de procesos y productos que se denomina plantilla de método, que incluye guías para la adaptación de los elementos que lo componen basado en los factores que conforman la situación del proyecto.

5. Construcción modular de métodos

Considerada por Harmsen como la solución más radical, este enfoque apunta a la creación de métodos a partir de módulos o fragmentos de métodos que puedan ser ensamblados de acuerdo a las circunstancias del proyecto. Los fragmentos se encuentran disponibles en un repositorio denominado Base de Métodos, codificados en un formato especial para su recuperación y su fácil ensamblado para construir así el método que mejor se adapte a la situación específica del proyecto.

La figura 10 muestra el espectro de los métodos situacionales.

2 Meta-modelado de métodos en la Ingeniería de Métodos Situacionales

Como pudo observarse en las definiciones comentadas en la Sección 1, un método contempla la dimensión del producto y la dimensión del proceso que se requiere para la creación del producto. Por ello, una de las premisas encontradas en la Ingeniería de Métodos Situacionales para el modelado de métodos es que se contemplan dos submodelos distintos como parte de un método, los cuales son el modelo de productos y el modelo de procesos. Cada uno de esos submodelos deben ser a su vez modelados, es decir, meta-modelados para realizar la descripción de un método en si.

Henderson y Ralyté indican que el uso de meta-modelos está recomendado en la Ingeniería de Métodos Situacionales (Henderson y Ralyté, 2010) y que la técnica de meta-modelado es central en esa disciplina. Los meta-modelos proveen medios para definir las reglas que afectan a un método o a un lenguaje de modelado en un nivel de abstracción más alto. En particular, son los meta-modelos los que permiten establecer los aspectos comunes entre distintos métodos y modelos conceptuales, tal como fue necesario hacer en el Capítulo II para comparar los distintas representaciones de los Procesos de Implementación.

Henderson y Ralyté reconocen en particular que el meta-modelado de métodos por parte de la ingeniería de métodos situacionales ha puesto especial énfasis en el modelado de procesos debido a que el modelado de productos se considera resuelto por los lenguajes de modelado como UML, y que se ha dejado de lado una dimensión también importante para comprender los métodos: la dimensión del equipo o las personas. Poco se ha trabajado con esta dimensión, y la literatura disponible sobre el tema es bastante limitada. Esto es un importante punto de mejora para la ingeniería de métodos situacionales.

A continuación se describen las distintas maneras en que se pueden representar los meta-modelos de producto y procesos, respectivamente.

2.1 Meta-modelado de Productos

El modelo de productos de un método describe las características de los productos que deben ser fabricados durante la ejecución del método, así como aquellos elementos que se consideran entradas requeridas para que los pasos indicados por el método puedan ser ejecutados.

El simple hecho de ser entidades fácilmente identificables hace que sea directo su modelado de la misma manera que se modelan los datos en un sistema de información. Un modelo de productos puede ser fácilmente realizado, como efectivamente se verá en el capítulo IV para describir el modelo de producto, usando un modelo de clases en UML. Otras notaciones, como el modelo E/R, podrían también colaborar en el meta-modelado de productos.

2.2 Meta-modelado de Procesos

Rolland expone que el modelado de procesos ha recibido menos atención que el modelado de productos (Rolland, 1997). El modelado de procesos por su parte cuenta con menos herramientas, especialmente cuando se propone la modularización de un proceso de manera que los componentes resultantes puedan ser seleccionados para su ejecución dependiendo de la situación.

Un proceso puede ser modelado usando modelos *orientados a la actividad*, modelos *orientados a producto* y modelos *orientados a la decisión*. Los modelos orientados a la actividad siguen la analogía de solución de problemas y proveen un marco para la gestión manual de proyectos. No obstante, su punto de vista lineal es inadecuado para métodos que soporten *backtracking*, reutilización de diseños previos e ingeniería paralela. Ejemplos de estos modelos son la mayor parte de los procesos de desarrollo de software. Los modelos de procesos orientados al producto representan el proceso de desarrollo a través de la evolución del producto. En tanto, los modelos de procesos orientados a la decisión, que no sólo involucran la actividad y los productos de la misma sino también las decisiones de diseño, facilitan el entendimiento de la intención del diseñador, por lo que promueve una mejor reutilización de los resultados (Rolland, 1997).

De acuerdo con Rolland, este enfoque orientado a la decisión ha sido aplicado en diversas oportunidades de manera acoplada con métodos situacionales en el contexto de la ingeniería de requisitos, siguiendo un enfoque contextual, que ha consistido en la organización de los *chunks* alrededor de la noción de contextos, concepto que involucra la relación entre la situación y la decisión tomada, y que tiene a su vez una relación unívoca con la intención subyacente (Rolland, 1997).

El meta-modelo de procesos de los componentes de métodos propuesto por Ralyté en (Ralyté, 2001) se basa en el concepto de directivas y está precisamente basado en el enfoque orientado a la decisión usado previamente por Rolland y que hace se basa en los conceptos de contexto (Pohl, 1994) y los mapas de procesos que permiten la navegación a través de las intenciones usando diferentes estrategias definidos por Rolland (Rolland, 1999).

2.2.1 Contextos

Uno de los primeros meta-modelos orientados a decisión fue el usado en el proyecto NATURE y documentado por Pohl en (Pohl et al, 1994). Dicho meta-modelo se basa en los conceptos básicos de: Situación, Intención, Contexto y Argumento.

- **Situación**: Consiste en la caracterización de la circunstancia sobre la cual se debe tomar una decisión. En el meta-modelo de Pohl está representado principalmente por una parte del producto que está siendo construido, y que debe ser transformado en la acción que se decida ejecutar.

- **Intención**: Representa la meta u objetivo que se desea lograr en una situación dada.

- **Contexto**: Consiste en un par ordenado de `<(Situación), Intención>` y representa el ámbito donde se toman las decisiones acerca de los próximos pasos a seguir. Al tomar una decisión y ejecutar una transformación sobre el producto (y por ende sobre la situación) se crea una nueva situación, que al relacionarla con la intención siguiente, se define el próximo contexto en el que se debe realizar una acción y tomar las decisiones que correspondan. Los contextos pueden ser de los siguientes tipos:

- **Contexto ejecutable:** Son aquellos contextos en los que se realiza una actividad o tarea simple, donde no se requiere tomar decisión alguna, por lo que puede ser definida de manera prescriptiva o puede ser incluso automatizada. La realización de un contexto ejecutable lleva implícita la ejecución de una acción o más acciones, que modifican la parte de producto que definen la situación del contexto (Pohl et al, 1994, Barrios, 2001).

- **Contexto de selección:** Se refiere a la parte de la definición del proceso donde deben tomarse decisiones. En un contexto de selección deben haber por lo menos dos alternativas, donde cada alternativa está representada por un contexto. Para elegir cualquiera de las alternativas, el adaptador del método a la situación se apoya en los argumentos que, a favor o en contra, se esgriman respecto a las circunstancias donde aplica o no la alternativa en evaluación.

- **Contexto plan:** Representa una estrategia compuesta para satisfacer una intención dada. Puede verse como un contexto conformado por una colección de contextos que deben ser ejecutados de acuerdo al orden establecido para lograr el objetivo.

- **Argumento:** Se relacionan con las alternativas presentadas en los contextos de selección para indicar los pros y los contras de cada alternativa de acuerdo a la situación dada.

Representación de los contextos

Los contextos ejecutables, dado el carácter atómico y de realización inmediata que tienen, pueden ser representados mediante una descripción textual, donde se mencionan las acciones a ser realizadas. Estas mencionadas acciones son actividades y tareas simples, por lo que su representación es preferible mantenerla tan simple como sea posible.

Para la representación de los contextos plan y selección, se hace uso de una notación especial, tal y como se describe en (Barrios, 2001). Un contexto plan, al considerarse como un conjunto de sub-contextos, muestra una lista de contextos unidos por una línea de conexión en ángulo recto, que puede ser leída como una conjunción Y. La notación formal utiliza el operador punto (**.**) para indicar la conjunción entre los sub-contextos del contexto plan. La figura 11 muestra un ejemplo de representación gráfica de un contexto plan, que se lee: *Contexto Plan consiste de la ejecución de los sub-contextos 1, 2* ***y*** *3*. La figura 20 muestra el mismo contexto plan de ejemplo siguiendo la notación formal.

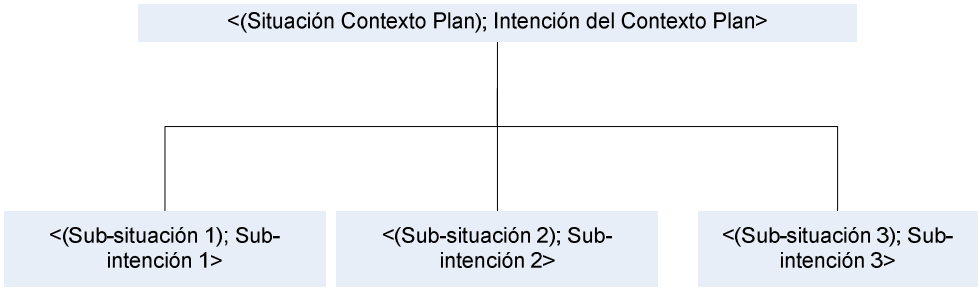


Figura 11. Representación gráfica de un contexto plan

```
<(Situación Contexto Plan); Intención del Contexto Plan) :=
                    <(Sub-situación 1); Sub-intención 1>.
                    <(Sub-situación 2); Sub-intención 2>.
                    <(Sub-situación 3); Sub-intención 3>
```

Figura 12. Notación formal de un contexto plan

Por su parte, los contextos de selección utilizan una notación similar, pero uniendo los nodos a través de líneas oblicuas que tienen asociadas los argumentos usados para decidir entre las alternativas disponibles en el contexto de selección representado. Esta notación representa una disyunción O y en notación formal usa el operador de unión (**U**). La figura 13 muestra un ejemplo de representación gráfica de un contexto de selección, el cual se lee: *En el Contexto Selección debe seleccionarse el sub-contexto 1 por el argumento A1 o el sub-contexto 2 por el argumento A2*. La figura 14 muestra la representación de mismo contexto de selección de acuerdo a la notación formal.

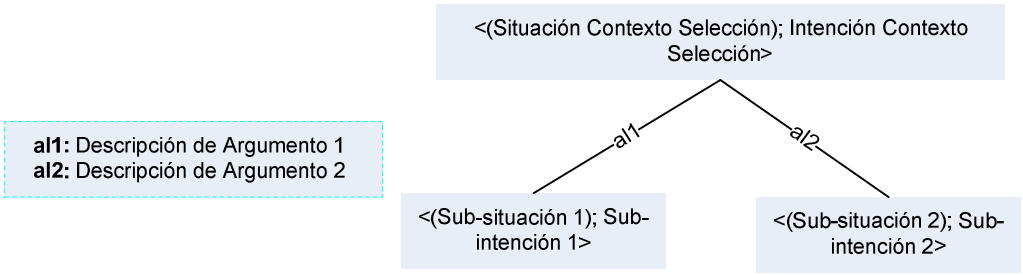


Figura 13. Representación gráfica de un contexto de selección


```
<(Situación Contexto Selección); Intención Contexto Selección) :=
                    <(Sub-situación 1); Sub-intención 1> U
                    <(Sub-situación 2); Sub-intención 2>
```

Figura 14. Notación formal de un contexto de selección

Los contextos plan y selección pueden ser ensamblados de manera tal que los sub-contextos de uno pueden ser de cualquiera de los tres tipos (ejecutable, plan o de selección). Así, podemos encontrarnos con contextos compuestos mixtos tal y como se muestra en la figura 15. En este caso, hacemos uso de uno de los contextos del modelo de procesos propuesto en el Capítulo V. La figura 16 nos muestra una representación algorítmica del mismo contexto, como una manera de explicar la notación.

Cada contexto compuesto (plan y/o de selección) tiene relacionado un ***grafo de precedencia***, el cual consiste de un grafo dirigido donde los nodos son los sub-contextos contenidos en el contexto plan y los arcos definen los encadenamientos posibles entre los mencionados contextos. Cada arco tiene relacionado un argumento que sirve como post-condición del sub-contexto origen y pre-condición del sub-contexto destino. La figura 17 muestra un ejemplo de grafo de precedencia.

En el Modelo de Procesos que se describe en el Capítulo V, se hace uso de la notación gráfica de los contextos plan, selección y compuestos para la descripción de los contextos de bajo nivel, es decir, los contextos que forman la ejecución de las intenciones de los mapas locales. Asimismo, para las directivas de los mapas se hace uso de la notación formal de los contextos de selección. No obstante, para mantener el nivel de detalle de

los contextos en un punto manejable, no se presentan ni la notación formal de los contextos de bajo nivel, ni sus respectivos grafos de precedencia.

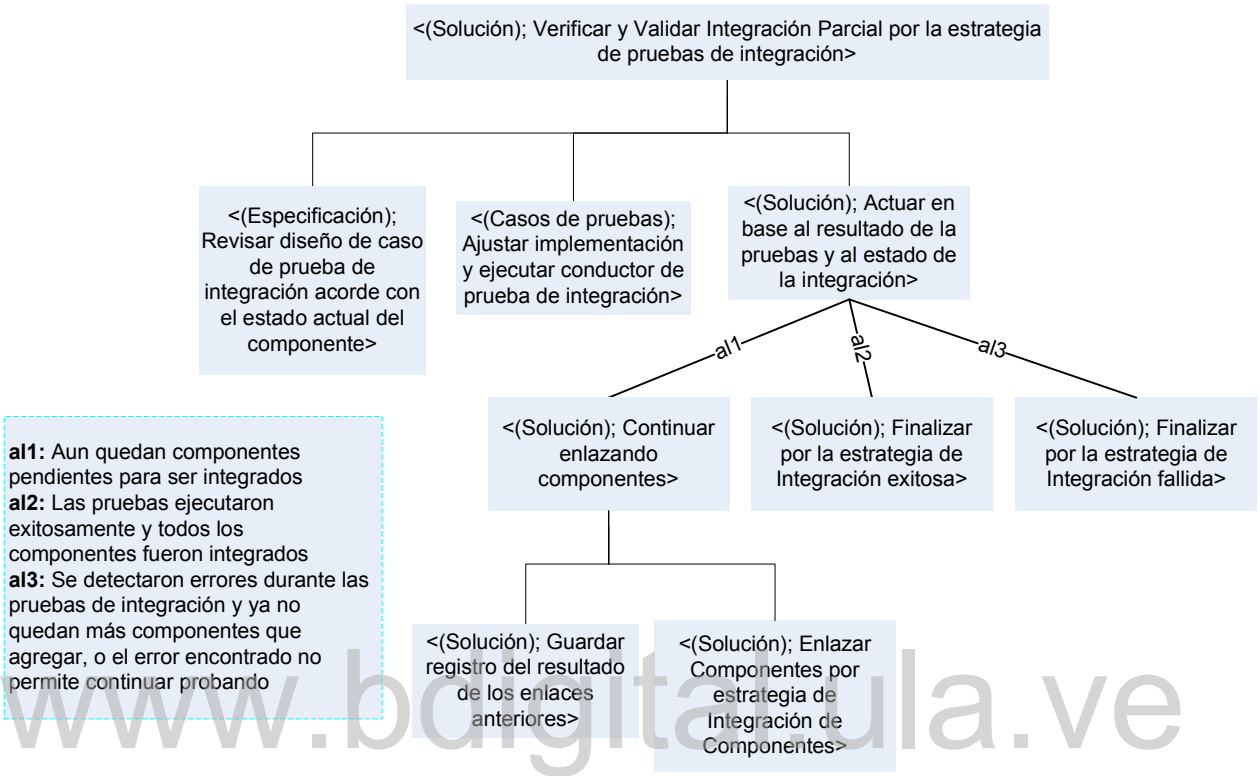


Figura 15. Ejemplo de un árbol de contextos compuestos entre plan y selección

```
Para Verificar y Validar la integración parcial
1.  Revisar el diseño del caso de prueba de integración
2.  Ajustar implementación y ejecutar conductor de prueba de integración
3.  Actuar en base al resultado de las pruebas de integración
        a.  Si aun quedan pendientes componentes para ser integrados, continuar enlazando los
            componentes
                i.  Guardar registro del resultado de los componentes
               ii.  Enlazar resto de los componentes
        b.  Si las pruebas se ejecutaron exitosamente y todos los componentes fueron
            integrados, finalizar con un resultado sin errores en la integración.
        c.  Si se detectaron errores durante las pruebas de integración y ya no quedan más
            componentes que agregar, o el error encontrado no permite continuar probado,
            finalizar el proceso indicando los errores encontrados en la integración.
4.  Fin
```

Figura 16. Representación algorítmica del contexto de la Figura 15

<(Solución); Verificar y Validar Integración Parcial por la estrategia de pruebas de integración>

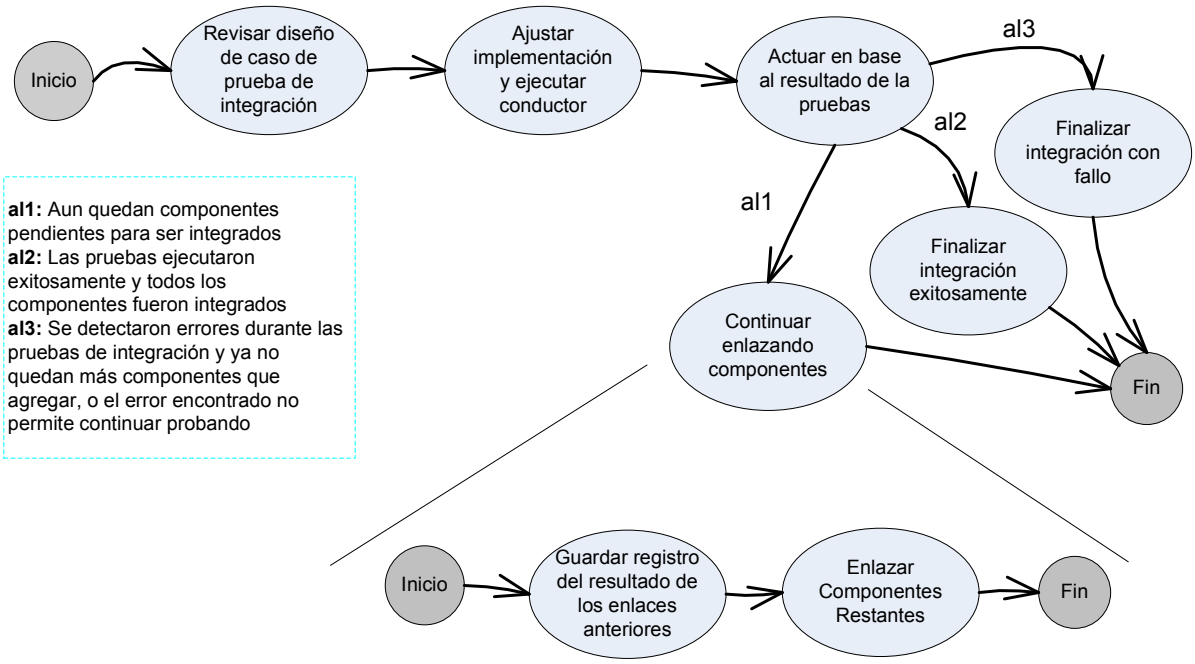


Figura 17. Grafo de precedencia del contexto de la Figura 15

2.2.2 Directivas

Una directiva se define como un conjunto de indicaciones sobre la manera de proceder para satisfacer un objetivo o ejecutar una actividad. Representa una sub-actividad de método y la parte esencial de un componente. Un componente de método, de acuerdo a Ralyté, es visto como una directiva asociada con una o más partes de producto necesarias para la ejecución de la actividad representada por la directiva (Ralyté, 2001).

De manera general, toda directiva posee una firma y un cuerpo. La firma caracteriza las condiciones sobre las cuales la directiva puede ser aplicada y el resultado de su ejecución, y se define como una dupla <*situación, intención*>. El cuerpo de la directiva, por su parte, define la actividad a realizar para satisfacer la intención que forma parte de la firma.

Una *intención* es un objetivo que se tiene en un momento dado en un proceso de ingeniería, la meta que se desea lograr al ejecutar una directiva. La *situación*, que representa la circunstancia al momento de buscar lograr una intención, es expuesta por Ralyté como una parte de producto en que es necesaria para lograr la intención (Ralyté, 2001).

Las directivas se clasifican en: Simples, Tácticas y Estratégicas.

Una directiva simple representa una actividad atómica, que puede estar asociada a una descripción textual o a una acción más o menos compleja a ejecutar. Puede ser informal o ejecutable. Las informales explican de manera textual el cómo proceder para obtener el producto de la directiva, en tanto que las ejecutables proponen una acción a ejecutar.

Una directiva táctica, por su parte, es una directiva compleja que usa una estructura de árbol para exponer sus sub-actividades. Estas directivas pueden mostrar un plan, un algoritmo estructurado de ejecución de la actividad, o bien puede mostrar una selección entre una o más alternativas. De acuerdo con (Barrios, 2001), una directiva táctica está definida por un contexto, tal y como fue definido por (Pohl, 1994).

Finalmente, una directiva estratégica es una estructura compuesta que usa una estructura de mapa de procesos, tal como se expuso en la sección 2.1, para exponer sus sub-actividades.

2.2.3 Mapa de Procesos

El mapa de procesos propone una vista estratégica de procesos que precisa que intención de procesos sigue a la ejecución de alguna estrategia en particular. Este modelo es un grafo dirigido donde las intenciones son representadas como nodos y las estrategias, que se definen como las diferentes maneras disponibles para lograr las intenciones, son representadas como arcos. Puede ser visto como una colección de secciones, que es la tupla formada por una intención origen, una intención destino y una estrategia. Cada sección tiene asociada una directiva denominada Directiva de Ejecución de Intención, que define como debe realizarse la intención destino a partir de la intención fuente siguiendo la estrategia asociada.

El mapa tiene dos intenciones que resaltan denominadas Inicio y Fin. El nodo Inicio permite comenzar la navegación en el mapa, y el nodo Fin permite finalizar la navegación. La figura 18 muestra un ejemplo de este mapa de proceso para directivas estratégicas. Allí puede observarse los nodos especiales de intención Inicio y Fin con una coloración distinta al resto de las intenciones.

La Directiva de Ejecución de Intención ayuda a la realización de una intención siguiendo una estrategia dada. Esta directiva puede ser representada con cualquiera de los tipos de directiva, sea simple, táctica o estratégica. Cada estrategia representada en el mapa de procesos por un arco entre dos intenciones tiene asociada su respectiva Directiva de Ejecución de Intención (DEI), la cual puede implicar la ejecución de una directiva estratégica que se traduce en la ejecución de un mapa de procesos de más bajo nivel o la ejecución de una directiva táctica, que se representa mediante un contexto plan, de selección o ejecutable.

Un mapa puede tener asociadas diversas estrategias de avance. Estas estrategias de avance buscan responder a las pregunta ¿Cómo seleccionar la siguiente intención con el fin de avanzar en el proceso? Las directivas de avance ayudan al ingeniero de aplicaciones a seleccionar las estrategias y las intenciones siguientes para avanzar en el modelo de procesos. Existen dos tipos de estrategias de avance: 1) las Directivas de Selección de Intención y 2) las Directivas de Selección de Estrategia. Estas directivas de selección se representan como contextos de selección, por lo que deben presentar argumentos para indicar que intención o estrategia seleccionar de acuerdo a la situación actual.

Las directivas de selección de intención determinan cuales son las intenciones que pueden seguir a la intención dada y ayuda a seleccionar entre alguna de ellas. En la representación gráfica de los mapas de proceso pueden observarse las Directivas de Selección de Intención denotadas dentro de aquellas intenciones de donde salen más de un arco y donde al menos un par de estos arcos posee intenciones destino diferentes. En el ejemplo de la figura 18, se observa la existencia de una directiva de selección de intención DSI1 dentro de la Intención 1, dado que desde ella se puede avanzar a las intenciones 2 y 3.

Las Directivas de Selección de Estrategia por su parte determina cuales son las estrategias que conectan a dos intenciones y ayuda a seleccionar entre alguna de ellas. En los mapas de proceso, estas directivas se representan asociadas a todas aquellas estrategias que tienen el mismo par ordenado (Intención Fuente,

Intención Destino). En el ejemplo de la figura 18, se observa como las estrategias E y F tienen asociada una Directiva de Selección de Estrategia 1, que indica los argumentos para definir cual de las estrategias seleccionar.

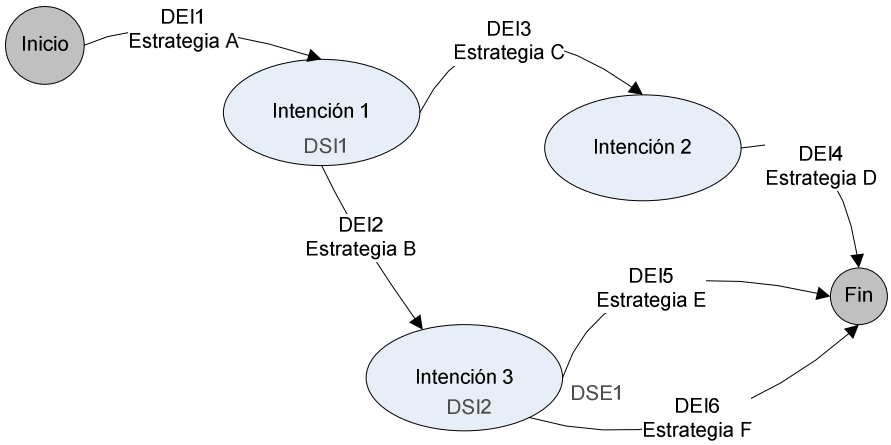


Figura 18. Ejemplo de un mapa de procesos.

2.3 Descripción de la Situación de un Proyecto

Como se ha visto, la situación es un elemento central en la ingeniería de métodos situacionales, y el principal determinante de diferenciación entre dos métodos distintos que cumplen sus objetivos en proyectos distintos. Con la caracterización de la situación se logran dos objetivos fundamentales en esta disciplina: 1) Entender los factores que marcan la diferencia en proyectos distintos y que hacen que la aplicación de un método o un fragmento de método sea recomendado o no en un proyecto dado; y 2) Crear los descriptores mediante los cuales los fragmentos de método pueden ser recuperados de una base de métodos una vez que se especifica las circunstancias específicas de un proyecto.

Entre la literatura disponible, se cuenta con los factores contextuales descritos por Van Slooten (Van Slooten, 1996), extraídos de una revisión de diversos proyectos estudiados en el campo de una organización bancaria. Los factores contextuales mencionados son:

- Compromiso de la administración con el proyecto
- Importancia del proyecto
- Impacto del proyecto
- Resistencia y conflicto
- Presión de tiempo
- Escasez de recursos humanos
- Escasez de medios
- Formalidad requerida en los procedimientos

- Conocimientos y experiencia del equipo de desarrollo
- Habilidades
- Tamaño
- Interacciones entre el sistema a desarrollar con otros sistemas
- Dependencia del proyecto con factores externos
- Claridad de las metas del proyecto
- Estabilidad de las metas del proyecto
- Nivel de innovación

Los factores contextuales de Van Slooten, a los que él denomina factores de contingencia, se encuentran en el ámbito del proyecto completo, afectando el contexto completo del mismo. De igual manera, se hace necesaria la definición de otros elementos de situación en ámbitos más puntuales.

En la literatura disponible acerca de los contextos (Pohl, 1994, Ralyté, 2001), se describe que la situación está formada a su vez por partes de productos y elementos del Modelo de Productos para el método en construcción. De esta manera, se establece un vínculo entre el estado actual de un producto con la situación, esperando que la ejecución del contexto que logra la intención en manos produzca un nuevo estado o situación que servirá de insumo al próximo contexto a seleccionar y ejecutar.

3 Enfoques y Técnicas de Construcción de Métodos Situacionales

De acuerdo con (Ralyte et al, 2003), existen las siguientes técnicas o enfoques para la creación de métodos en la ingeniería de métodos: el enfoque basado en ensamblaje, el enfoque basado en el uso de patrones, y el enfoque basado en paradigma. Adicionalmente, (Järvi et al, 2007) proponen un enfoque de ingeniería de métodos basada al desarrollador. A continuación se revisan de manera general cada una de estas técnicas:

3.1 Ingeniería de métodos basada en ensamblaje

El enfoque basado en ensamblaje representa la construcción en tiempo real del método basado en la caracterización de la situación para con ellos seleccionar un fragmento de método, que se toma de métodos existentes para satisfacer la situación actual dada. De acuerdo con Ralyte, apunta a la construcción de un método '*on-the-fly*' para ajustarse lo mejor posible a la situación del proyecto en manos (Mirbel y Ralyté, 2006). Para esto, hace uso de componentes de método reusables, denominados *chunks*, que se seleccionan desde un repositorio donde se encuentran de acuerdo a los requerimientos del proyecto, ensamblándolos entre si.

La construcción de métodos basada en ensamblaje sigue un paradigma tradicional de ingeniería, donde sus pasos claves consisten en 1) especificar los requerimientos del método, 2) seleccionar los *chunks* de método apropiados y 3) ensamblar los *chunks* seleccionados. (Mirbel y Ralyté, 2006).

La especificación de requisitos es realizada siguiendo las estrategias orientadas a la intención, apropiada para la adaptación de métodos u orientadas al proceso, estrategia esta que se ajusta a la creación de un método desde el inicio. La estrategia orientada a la intención consiste en la identificación de las intenciones que

completarían, mejorarían o acotarían el método a adaptar. Por su parte, la estrategia orientada al proceso, busca no sólo la identificación de las intenciones a adaptar sino todas las intenciones requeridas que deben ser satisfechas por el nuevo método a crear.

La selección de los *chunks* apropiados de método consiste en la recuperación de los mismos desde la base de métodos correspondiendo a partir de los requisitos previamente especificados. Las estrategias de descomposición, agregación y refinamiento ayudan en la selección del *chunk* candidato analizando más en profundidad si se corresponde a los requisitos. Finalmente, los *chunks* seleccionados deben ser ensamblados para conformar un nuevo método o ampliar uno que ya exista.

3.2 Ingeniería de métodos basada en extensión

El enfoque basado en extensión se corresponde con el objetivo de adaptar localmente un método a la contingencia del proyecto que se tenga entre manos (Ralyte, Deneckère y Rolland, 2003). Este enfoque guía al ingeniero de métodos al proveer patrones de extensión que ayudan a identificar las situaciones de extensión típica y provee consejos para desarrollar la extensión requerida

Esta técnica puede ser ejecutada de dos maneras distintas: a través de la estrategia de mezcla de patrones o el uso de algún conocimiento genérico relacionado con el dominio para el cual la extensión es hecha a través de la selección de un meta-patrón y de la extensión de un método con la estrategia basada en patrón.

3.3 Ingeniería de métodos basada en paradigma

Este enfoque usa el meta-modelado como su técnica subyacente de ingeniería de métodos (Ralyte, Deneckère y Rolland, 2003). Es la técnica de construcción más genérica entre las que fueron revisadas. Consiste en la representación de modelos de producto y de procesos como instanciación del mismo meta-modelo. Este enfoque es también conocido como meta-modelización.

El meta-modelado es conocido como una técnica para capturar el conocimiento acerca de los métodos. Sienta las bases para el entendimiento, comparación, evaluación e ingeniería de métodos. Uno de los resultados obtenidos por la comunidad de meta-modelado es la definición de cualquier método como compuesto de un modelo de producto y un modelo de procesos. La construcción de métodos siguiendo la técnica de meta-modelado se centra en la definición de estos dos modelos.

La construcción del modelo de productos depende de la meta de ingeniería de métodos. Estas metas pueden ser el construir un método: 1) por aumentar o disminuir el nivel de abstracción de un modelo dado; 2) por instanciar un meta-modelo seleccionado; 3) por adaptar un meta-modelo a algunas circunstancias específicas y 4) por adaptar un modelo.

3.4 Ingeniería de métodos orientada al desarrollador

Las técnicas presentadas anteriormente en esta sección comparten la característica de separar el diseño del método de su uso, en términos del momento en que ocurre, así como de los roles de quienes lo llevan a cabo; es decir, que los métodos son previa y exclusivamente diseñados y adaptados por ingenieros de método antes de su uso y aplicación en un proyecto. Algunos autores mencionan la importancia de buscar estrategias complementarias para esta manera de adaptar métodos a proyectos. Aaen (Aaen, 2003) argumenta que cuando la responsabilidad del diseño de procesos recae más en especialistas de procesos que en quienes los usan y aplican, los procesos resultantes tienen poca interacción con los proyectos de software reales, su aplicación

podría convertirse en un ritual por tradición y poco reflexivo y puede conllevar a la osificación y rigidez de los procesos. Järvi, Hakonen y Mäkilä (Järvi et al, 2007) exponen que la separación del diseño de los procesos de su uso, la externalización del conocimiento del proceso y la estructuración de los módulos de procesos para formar un sistema coherente, representan una única estrategia posible para hacer frente a la complejidad y la incertidumbre de los proyectos de software actuales.

Järvi, Hakonen y Mäkilä proponen un enfoque de la ingeniería de métodos situacionales que complementa la manera tradicional de diseñar y adaptar métodos, donde se incluye un diseño de métodos orientado al desarrollador para permitir la adaptación de los métodos incluso durante la ejecución del proyecto. Este enfoque presenta cuatro niveles de ingeniería de métodos: 1) Administración de la biblioteca de métodos, en la que se mantienen métodos de manera modular para facilitar el uso y reuso de estos métodos en los otros tres niveles y cuya gestión es responsabilidad de los ingenieros de método y la alta administración. 2) Diseño de métodos para proyectos específicos, donde se describe un método para una situación en particular con la intención de imponer control sobre los desarrolladores pero dejando opciones abiertas cuando la necesidad de adaptación pueda ser anticipada. 3) Adaptación de métodos de acuerdo a la situación, bajo la responsabilidad de los usuarios de los métodos, para satisfacer las necesidades específicas que se tengan. Y finalmente, 4) Construcción *on-the-fly* de métodos, como respuesta a situaciones no anticipadas, donde en la construcción del fragmento se comunica el plan para hacer frente a la situación y se documenta el mismo para su posterior uso en las actividades de mejora de procesos.

Este tipo de construcción *on-the-fly* de métodos se enfoca más en la generación de planes para responder a las situaciones inesperadas que en la generación de guías documentadas. Cuando se complementa con la apropiada reutilización de métodos de la biblioteca de fragmentos, esta construcción de métodos se hace mucho más rápida.

4 Definición de Estrategia de Modelado del Proceso de Implementación desde la perspectiva de métodos situacionales

Cómo se ha expuesto en el capítulo I, es uno de los objetivos de este trabajo el modelado de un proceso de implementación ágil y disciplinado usando el enfoque de la ingeniería de métodos situacionales. La estrategia a ser aplicada para la definición de los fragmentos de método involucra en una primera instancia la definición de un modelo de productos asociados a la Implementación de Software. Esta definición se realiza en el Capítulo IV del presente trabajo, donde los fragmentos de producto son descritos usando la notación de un diagrama de clases UML.

Adicionalmente, se requiere la definición de los fragmentos de proceso, que se realizará de manera multinivel haciendo uso de los conceptos de mapas (Rolland, 1999) y de directivas (Ralyté, 2001). Se consideran las intenciones de nivel alto y medio como directivas estratégicas, de acuerdo a la clasificación de Ralyté, por lo que se describen en el Capítulo V usando la notación de mapas.

En términos generales, se definen los siguientes principios que se siguen en la definición del Modelo de Procesos en el Capítulo V:

- Se inicia con una directiva estratégica global asociada al Proceso de Implementación completo, con una intención y situación definida. Esta directiva global se considera de alto nivel.

- Las directivas estratégicas del nivel alto y medio se representan mediante Mapas de Proceso. Las Directivas de Selección de Intención y Selección de estrategia de estos mapas se representarán como directivas tácticas en forma de contextos de selección.

- Las Directivas de Ejecución de Intención del mapa de nivel alto serán a su vez modeladas como directivas estratégicas, representadas en Mapas de Proceso, que se consideraran de nivel medio.

- Las Directivas de Ejecución de Intención de los mapas de nivel medio serán modeladas como directivas tácticas, en forma de contextos plan, selección, ejecutable o una composición de ellos.

- Los contextos plan y selección se modelan siguiendo la representación gráfica y una descripción textual de lo que allí se desea representar. En el alcance de este trabajo, no se incluyen las notaciones formales de estos contextos ni sus respectivos grafos de precedencia.

- Los contextos ejecutables son descritos utilizando un lenguaje natural, con párrafos cortos que explican el trabajo que en ellos debe hacerse. Estas descripciones se omiten en los contextos ejecutables que forman parte de contextos plan o de selección y se consideran como explicados en las descripciones textuales de los contextos que los contienen.

El diagrama de la figura 19 define el meta-modelo de procesos que será usado para la descripción del Modelo de Procesos en el Capítulo V, de acuerdo a los principios ya listados.

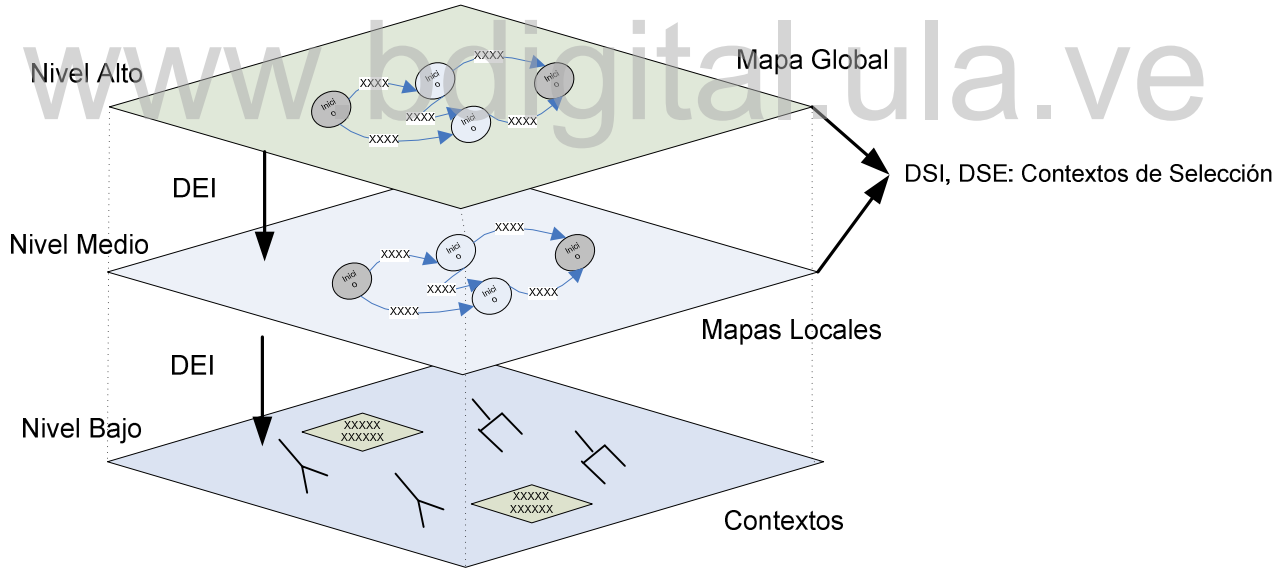


Figura 19. Diagrama de los niveles del meta-modelo de procesos usado en este trabajo

Si bien se encuentra un gran valor en la representación de fragmentos situacionales para a la integración de prácticas de métodos y modelos ágiles y disciplinados, un aspecto que debe destacarse es la complejidad añadida en cuanto a la lectura del modelo de procesos debido a la notación propia de la ingeniería de métodos situacionales entre mapas, directivas y contextos, lo cual limita la utilidad del método resultante una vez que se cuenta con los valores actuales (instancias) de los factores contextuales considerados como parte de la situación. Para aminorar tal dificultad, se procede a la descripción tabular de la relación entre intenciones, estrategias y directivas con sus respectivos argumentos para cada uno de los mapas globales y locales, buscando con ello facilitar la lectura de los fragmentos de procesos descritos en el Capítulo V. No obstante, se

hace evidente que la lectura de estos fragmentos puede ser facilitada de gran manera contando con las herramientas automatizadas apropiadas para tal tarea.

4.1 Caracterización de la Situación

Debido a las características prácticas de las actividades llevadas a cabo dentro del Proceso de Implementación de Software, no todos los factores contextuales descritos en la sección 2.3 tienen impacto en la selección de los fragmentos de método, dado su origen en el ámbito organizacional y no técnico. Las situaciones de los contextos descritos en el Modelo de Procesos del Capítulo V son formados en su mayoría por partes del Modelo de Producto para su transformación por parte del contexto. Sin embargo, algunos factores contextuales tienen especial impacto a la hora de seleccionar un fragmento de método dentro del Proceso de Implementación que se tiene como objetivo modelar. Estos factores contextuales a considerar son:

- **Experiencia del equipo**: Consiste en el grado de experiencia disponible en los miembros del equipo, y que está asociado a la cantidad y calidad de experiencias previas en proyectos similares o con elementos singulares que se asemejan a las características del proyecto en manos. Puede calificarse en los siguientes valores:
    - **Baja:** El equipo está formado en su totalidad por personas con poca o ninguna experiencia previa, ni siquiera en proyectos de otro tipo.
    - **Baja a media:** El equipo está conformado parcialmente por personas con poca experiencia, mientras algunos miembros poseen algo de experiencia en proyectos similares.
    - **Media:** El equipo de desarrollo está equilibrado entre personas con amplia experiencia y personas con poca experiencia, o todos los miembros del equipo poseen algo de experiencia en proyectos similares.
    - **Media a alta:** El equipo de desarrollo posee un buen nivel de experticia, donde todos los miembros poseen experiencias previas en proyectos similares, mientras algunos miembros pueden ser calificados como expertos en el tipo de proyectos o en la tecnología a ser usada.
    - **Alta:** El equipo está formado en su gran mayoría por expertos en el área y en la tecnología y con amplia experiencia en proyectos similares.
- **Habilidades disponibles en el equipo**: Consiste en el conocimiento y experiencia en la aplicación de técnicas o prácticas o en el uso de una tecnología de soporte para aumentar la productividad de la implementación de componentes de software. Entre estas habilidades podemos encontrar: Conocimiento de técnicas de desarrollo orientado a pruebas, uso y manejo de sistemas de control de versiones, entendimiento de los conceptos asociados a la integración continua, etc.
- **Tecnología sobre la que se va a implementar:** Dependiendo de la tecnología a usar para la implementación de los componentes de software, se puede contar con ciertas y determinadas prácticas y herramientas para aumentar la productividad de la implementación. Muchos lenguajes de programación y tecnologías usadas en la actualidad ponen a disposición herramientas con las que no se contaba hace veinte años, por lo que si el componente a desarrollar debe ser codificado en lenguajes de programación antiguos como COBOL, la disponibilidad de herramientas de productividad es reducida.

- **Disponibilidad de herramientas automatizadas de soporte:** Este factor contextual está asociado al factor mencionado anteriormente, ya que algunas herramientas pueden o no estar disponibles para ciertas tecnologías. Sin embargo, se observa frecuentemente el caso donde la tecnología permite el uso de herramientas automatizadas de soporte pero por razones diversas la herramienta sigue sin estar disponible o no ha sido usada aun estando disponible.

5 Conclusiones de este capítulo

La ingeniería de métodos situacionales parte de la premisa de la no existencia de un método universal que pueda adaptarse a las distintas circunstancias de cada proyecto, y promueve la creación de métodos que se adapten y se ajusten a la situación del proyecto donde será aplicado.

El meta-modelado cumple una función central en la ingeniería de métodos situacionales porque permite definir las reglas más generales en un nivel de abstracción más alto. Con ello se logra que métodos que pudiesen estar cubriendo necesidades distintas en situaciones completamente diferentes, mantengan cierto nivel de congruencia y relación en aquellos aspectos en que son similares.

El modelado de un método requiere de un meta-modelo de productos y un meta-modelo de procesos. Un modelo adicional que no ha sido descrito en la literatura evaluada es el de la dimensión del equipo de desarrollo y las actividades asociadas a cada rol. Esto es un importante punto de mejora para la ingeniería de métodos situacionales.

El modelado de los fragmentos del Proceso de Implementación se apoya principalmente en los Diagramas de Clases en UML para el Modelo de Productos en el Capítulo IV, y se hace uso de la notación de mapas, de directivas y de contextos para el Modelo de Procesos en el Capítulo V.

La caracterización de la situación asociada a los fragmentos del Proceso de Implementación está principalmente basada en partes de producto provenientes del Modelo de Productos, con la inclusión de algunos factores contextuales que están estrechamente asociados a las actividades prácticas propias del Proceso de Implementación de Software.

CAPITULO IV: Modelo de Productos de Implementación de Software

En este capítulo se presenta el modelo de productos propuesto como base para la construcción del método situacional que represente el Proceso de Implementación con características ágiles y disciplinadas. Dicho modelo de productos contempla no sólo las salidas esperadas de un Proceso de Implementación de software sino que también se ocupa de la caracterización de aquellos conceptos y entidades que deben ser provistos al proceso como entradas y de los productos de trabajo intermedio que podrían ser generados. Los productos definidos en este modelo tienen como fuentes principales los estándares y métodos revisados en el Capítulo II y es un subconjunto del modelo integrado presentado en el mismo capítulo; para el modelado de estos conceptos se hará uso del Diagrama de Clases en notación UML (OMG, 2009).

Por tratarse de un modelo de productos para definir fragmentos situacionales de método, algunos de los productos que se definen en las secciones siguientes pueden ser representados de maneras distintas que son dependientes de la estrategia seleccionada de acuerdo a la situación dada del proyecto. El modelo que se describe en este capítulo representa este tipo de situación haciendo uso de la notación de Clase Abstracta, en tanto que las instancias específicas se representaron usando la notación de Clase Concreta.

Un reto importante de la definición de estos productos consiste en la integración de la terminología usada por los distintos estándares revisados en el Capítulo II, lo cual tiene fuertes implicaciones en el entendimiento de este modelo que trata de conciliar los puntos de vistas de diversos representantes de cada perspectiva. Para minimizar el impacto en este sentido, el Apéndice A presenta el glosario de los términos usados tanto en el Modelo de Productos como en el Modelo de Procesos, descritos en este capítulo y el Capítulo V, respectivamente.

La definición de nuestro modelo de productos procura describir la esencia de estos conceptos más que la forma o representación de los mismos. Un ejemplo claro está en los modelos que puedan hacerse de un producto de software. Regularmente se espera que tales modelos se vean expresados en forma de documentos, que típicamente se convierten en elementos de configuración y que deben ser actualizados constantemente, lo cual crea una sobrecarga de esfuerzo nada despreciable. La propuesta de minimización de documentos, o el uso de las prácticas recomendadas por Ambler en (Ambler, 2002) acerca de modelado y documentación ágil, no proponen en modo alguno la inexistencia de tales insumos, sino su trato como modelos antes que como documentos. Por lo expuesto anteriormente, el modelo de productos que se presenta tiene entre sus principios de diseño el no hacer referencia explícita a alguna representación específica de algún elemento en particular (ej. Documento de diseño, Documento de especificación de requisitos). Esto es importante en especial si puede existir una representación que promueva de mejor manera la agilidad y la eficiencia durante la implementación.

Para facilitar el entendimiento del modelo de productos propuestos, se dividen los conceptos o entidades por su rol o clasificación en el Proceso de Implementación de Software, tal como se describió en la sección 1 del capítulo II. De esta manera, se analizarán primero por separado y luego en conjunto, las entradas, productos de trabajo intermedios y salidas del proceso. Adicionalmente, y por ser importantes para la perspectiva integrada ágil y disciplinada, se detallarán en el modelo de productos las herramientas que soportan nuestro Proceso de Implementación.

1 Entradas del Proceso de Implementación

La figura 20 presenta el conjunto de entradas del modelo de productos de nuestro Proceso de Implementación. Como elemento central de las entradas requeridas se tiene la ***Especificación del Sistema***, que representa el

resultado de los procesos previos a la Implementación del Software (Análisis y Diseño del Sistema), que contiene información sobre la ***Solución*** que fue recabada en dichos procesos. Es por esto que una representación mínima de la especificación de la solución debe contener la ***Especificación de Requisitos*** y la ***Especificación Arquitectónica***. La ***Especificación de Requisitos*** consiste en el compendio de los requisitos que deben ser satisfechos por la solución. Su representación mínima debe contener los requisitos funcionales y no funcionales de la solución, y por cada requisito debe contemplar su descripción, los flujos base y alternativos, criterios de aceptación y la prioridad que tiene para el cliente y los involucrados principales.

La ***Especificación Arquitectónica*** consiste en la descripción de alto nivel del diseño de la solución, también conocida como Diseño Arquitectónico. De acuerdo al estándar ISO/IEC 42010:2007 de prácticas recomendadas para la descripción arquitectónica (ISO/IEC, 2007), las vistas incluidas en la descripción de una arquitectura son dependientes de las necesidades de los interesados e involucrados en un proyecto y por lo tanto ninguna vista es considerada como indispensable. No obstante, dado que si son requeridas aquellas vistas que satisfagan las necesidades e inquietudes de los interesados e involucrados, este modelo arquitectónico siempre será provisto para el Proceso de Implementación.

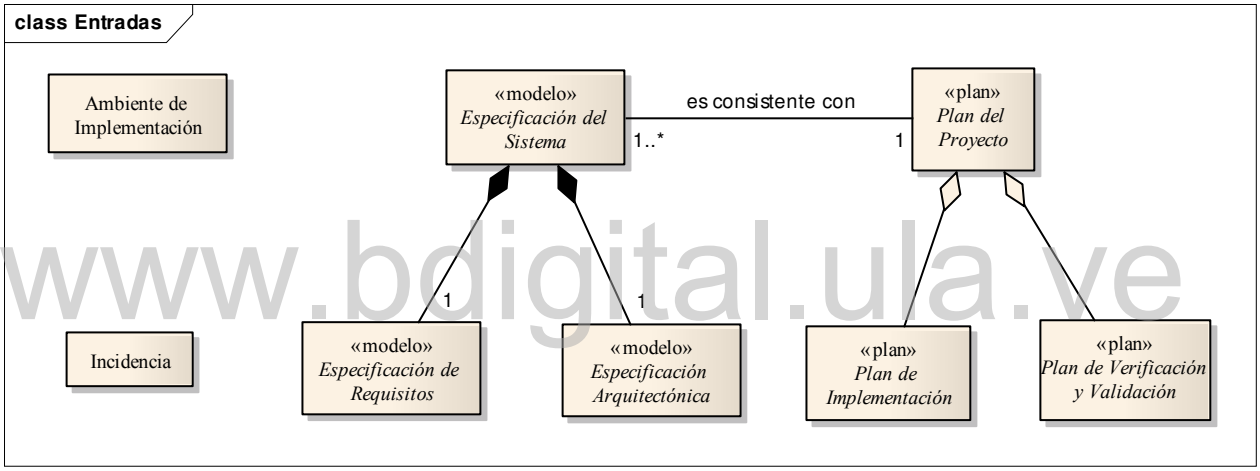


Figura 20. Diagrama de Clases de Entradas del Proceso de Implementación

El ***Plan del Proyecto*** se refiere al plan maestro que contempla la planificación de las distintas actividades que deben ser realizadas durante el desarrollo del software. Este plan puede presentarse como un documento conteniendo una planificación exhaustiva con alto nivel de detalle, o bien como un plan de alto nivel para las fases de alto nivel del proyecto de desarrollo, entre las cuales se encontraría el Proceso de Implementación. Para este proceso específico, se requiere el detalle de los requisitos que deben ser satisfechos y/o los componentes que deben ser implementados, siempre manteniendo la consistencia con las prioridades que se especifican en los requisitos. Este detalle de la planificación acerca de los elementos que deben ser implementados estaría contenido en una parte del plan total que se denomina ***Plan de Implementación***.

Similar al Plan de Implementación se tiene el ***Plan de Verificación y Validación***, el cual debe contemplar la planificación de las actividades de verificación y validación, que se espera sean ejecutadas asociadas a los productos de trabajo intermedios y finales del Proceso de Implementación. Entre las actividades y tareas que el Plan de Verificación y Validación contempla se encuentran el desarrollo de los ***Casos de Pruebas Unitarias*** y de ***Integración*** y la posterior ejecución de las actividades asociadas a dichas pruebas. Debe incluir también la planificación para la ejecución de revisiones de pares.

Una entrada opcional, con el cual se pueden indicar ajustes menores que requieran los componentes debido a errores durante la validación con el cliente, son las ***Incidencias*** o *Issues*. Estas incidencias, que pueden ser manejadas de diversas maneras, están asociadas a uno o más componente en particular que deben ser revisados y ajustados.

Finalmente, el Proceso de Implementación requiere de la preparación previa de un ***Ambiente de Implementación***. Este ambiente representa el conjunto de recursos que se utilizan para llevar a cabo la implementación del software, como los recursos de hardware y software de soporte que servirán de herramientas los ambientes de desarrollo necesarios para los programadores y el marco de convenciones del equipo de desarrollo seguidas durante toda la implementación. Se considera que este ambiente de implementación evoluciona a medida que el Proceso de Implementación transcurra. El criterio que priva en este caso es que al momento de iniciarse el Proceso de Implementación se disponga de los recursos necesarios para las primeras actividades definidas en el Plan de Implementación y que puedan resultar en obstáculos para la ejecución del proceso. Ejemplo de esto son los ambientes locales de desarrollo, sin los cuales los desarrolladores pueden aportar muy poco valor. Los elementos mínimos que debe contemplar un Ambiente de Implementación son: ambiente de desarrollo configurado para cada desarrollador, plataforma de herramientas de soporte instaladas, convenciones de desarrollo corporativas o requeridas por el cliente para el proyecto al alcance del equipo de desarrollo.

2 Salidas del Proceso de Implementación

Se denominan salidas a aquellas entidades resultantes de la ejecución del Proceso de Implementación. El modelo de Salidas del Proceso de Implementación aparece representado en el diagrama de clases mostrado en la figura 21. Como centro de las salidas del Proceso de Implementación se tiene la ***Solución***, que consiste en el sistema completo con elementos de hardware, software y documentación, que es desarrollado y es el fin del proyecto que enmarca el proceso entero. Un componente importante de la Solución es el ***Producto de Software***, que es el conjunto de programas de computadora o componentes de software que satisface como un todo los requisitos del cliente. El ***Producto de Software*** es el resultado de integrar los distintos ***Componentes de Software*** que son diseñados, codificados y verificados durante el Proceso de Implementación.

La Solución requiere de un esfuerzo posterior al Proceso de Implementación descrito en este trabajo antes de poder considerarse completa y que está representado por la validación del producto y el despliegue y puesta en producción del mismo. La Solución no sólo incluye el software que satisface los requisitos del cliente, sino que incluye también elementos adicionales como procedimientos manuales que requieran ser instaurados, adiestramientos que sea necesario llevar a cabo, así como la documentación requerida para la operación del ***Producto de Software*** y algún otro documento que haya sido explícitamente requerido por el cliente.

En este trabajo se considera que un ***Componente de Software*** es una unidad de software que engloba una responsabilidad específica en el contexto de la arquitectura planteada para el ***Producto de Software*** y que satisface o colabora en la satisfacción de uno o más requisitos del cliente. En programación orientada a objetos, un componente estaría representado por una clase, que puede efectivamente ser verificada utilizando pruebas unitarias. Los componentes de software pueden ser de varios tipos, como ***Componentes de Interfaz de Usuario***, que son usados en la interacción del usuario con el software y a través de ellos se hace disponible la funcionalidad al usuario; también pueden ser ***Componentes de Lógica de Dominio***, con los cuales bien se representan las entidades o conceptos clave del sistema de negocios que se busca modelar dentro de la ***Solución***, o se implementan las reglas propias del mismo sistema de negocios; así también pueden ser

Componentes de Base de Datos, los cuales se orientan al almacenamiento estructurado de los datos que son procesados por el producto de software.

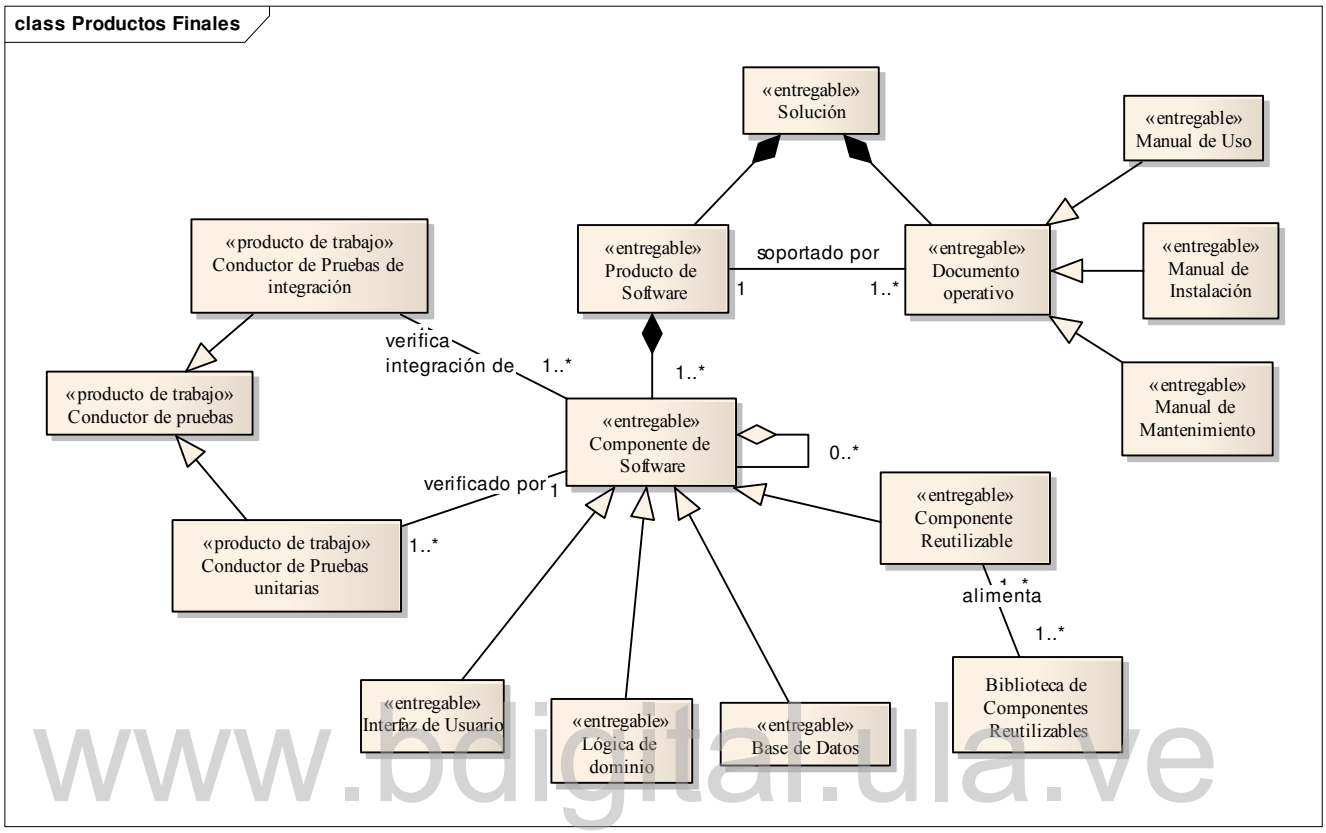


Figura 21. Diagrama de Clases de Salidas del Proceso de Implementación

Transversal e independiente a esta clasificación, un componente puede ser un ***Componente Reutilizable***, lo que representa que puede ser reutilizado en un contexto distinto a la aplicación actual. Los componentes reutilizables usualmente son dispuestos en bibliotecas de componentes que pueden ser consultadas para ubicar algún componente al que se pueda delegar una responsabilidad. Cuando se observa la oportunidad de hacer reutilizable un componente diseñado para un proyecto en particular, debe aprovecharse y disponerse en estas bibliotecas de componentes reutilizables, lo que crea una relación simbiótica entre estas bibliotecas y los proyectos de desarrollo de software.

Un aspecto a resaltar aquí es la importancia que se da a los ***Conductores de pruebas***, sean unitarias o de integración, y que se incluyen como salidas del proceso. Un ***Conductor de pruebas*** es un programa que se encarga de verificar que un componente o un conjunto de ellos realiza una operación de acuerdo a las especificaciones provistas. Estos conductores son de gran valor para la verificación en las actividades de codificación e integración de componentes, razón primordial por la que son generados durante el Proceso de Implementación. Sin embargo, estos conductores pueden servir como herramientas útiles durante las actividades de verificación y validación que se tengan a lugar realizar luego del ***Proceso de Implementación***, lo que justifica que se consideren como salida del proceso.

Para culminar, otros elementos de la ***Solución*** que debe ser generado por el equipo de desarrollo se refieren a los ***Documentos Operativos***, que contempla aquellos manuales necesarios, como su nombre lo indica, para la

operación, instalación y mantenimiento de los componentes. Los documentos operativos o manuales fueron seleccionados de acuerdo de lo recomendado en (Montilva, Barrios y Rivero, 2008). Por ello entre estos documentos operativos se cuenta con el ***Manual de Uso***, dirigido a los usuarios de la aplicación y que describe la funcionalidad de la aplicación; ***Manual de Instalación***, que explica como llevar a cabo la instalación de los componentes de la ***Solución*** en la plataforma de operación establecida; y ***Manual de Mantenimiento***, dirigido al equipo que realizará las labores de mantenimiento de la aplicación y que describe los aspectos de diseño e implementación de la aplicación. En este caso, la presentación en forma de documento si es requerida, por ser uno de los elementos mínimos que se requiere para usar, dar soporte y mantenimiento al ***Producto de Software***. Sin embargo, y con miras a mantener la agilidad y la transparencia con el cliente, estos documentos deben ser agregados a la Especificación de requisitos como uno más de ellos y así ser priorizados de la misma manera que se hace con cualquier otro requisito.

3 Productos de Trabajo Intermedios del Proceso de Implementación

Se considera un producto de trabajo intermedio aquel elemento que resulta de la realización de una tarea o actividad puntual dentro del proceso que se describe y que es necesario para el desarrollo de un producto final del proceso. Un producto de trabajo intermedio puede o no ser un entregable en si como parte del Proceso de Implementación. En la figura 22 se observa el Modelo de Productos de Trabajo Intermedios del Proceso de Implementación. Sentando la infraestructura sobre la que se construyen los productos intermedios y finales, se tiene el ***Ambiente de Implementación***, verificando que la ***Plataforma de Desarrollo*** y las ***Convenciones de Desarrollo*** se encuentren en un nivel mínimo para soporte de las actividades del proceso. La ***Plataforma de Desarrollo*** consiste en la capa de sistemas hardware y software que dan soporte a las actividades de desarrollo. Las ***Convenciones de Desarrollo*** consisten en los acuerdos explícitos realizados previamente al inicio de la codificación y que establecen un marco común para decisiones que de otra manera seguramente cada desarrollador distinto aproximaría con decisiones distintas. Estas convenciones de desarrollo contemplan uno o más ***Estándares de Construcción***, que tienen como objeto específico promover la consistencia a lo largo del diseño detallado y la codificación de los componentes. Estos estándares, y así con otras convenciones de desarrollo pueden tener como origen un requisito o restricción del cliente, o pueden ser impulsados como convenciones comunes a todos los proyectos de la organización que realiza el desarrollo. De igual manera, las convenciones de desarrollo involucran definición de elementos como la ***Biblioteca de Componentes Reutilizables***, que será de gran importancia para la determinación de las oportunidades de reutilización durante el aprovisionamiento de los componentes.

Otro producto de trabajo intermedio que es realizado durante la implementación del software es el ***Modelo de Diseño Detallado***. Dicho modelo consiste en aterrizar al nivel más bajo posible – a las actividades de creación de código fuente – las especificaciones provistas por los requisitos y la representación arquitectónica. El principal objetivo de este modelo es el entendimiento y la comunicación entre los desarrolladores respecto a una manera apropiada, o la más apropiada para el caso, de implementar lo que se le solicita. Bajo esta perspectiva, este modelo no requiere convertirse en un documento, salvo que sea exigencia explícita. No obstante, se recomienda evaluar concienzudamente si la exigencia amerita serlo, puesto que una de las actividades que más genera sobrecarga de esfuerzo es mantener actualizado con el código fuente un documento de diseño detallado, y tener un documento de diseño detallado desactualizado puede ser peor a no tener ninguno (Ambler, 2002).

Se denomina ***Elemento de Trabajo*** a la especificación de actividades que pueda asignarse de manera sencilla a un desarrollador, que tenga suficiente independencia de otras actividades como para poder ser verificable su correcta implementación. Es la unidad básica de asignación de actividades de construcción de componentes.

Un ***Elemento de Trabajo*** debe tener una clara y concisa descripción de la tarea o actividad que debe realizarse, debe ser probable y debe indicar la forma en que debe ser probado. Estos elementos pueden ser asignados de manera centralizada o pueden ser seleccionados para su ejecución de manera distribuida (cada desarrollador decide qué elemento asignarse). Esta última estrategia es ampliamente usada en los métodos ágiles, como XP (Beck, 2000).

La ***Interfaz del Componente*** de software es el límite compartido a través del cual la información fluye (*IEEE* Standards Board, 1990). Contempla la caracterización de las operaciones que pueden ser llamadas a ejecutar desde otros componentes. Tal interfaz puede o no haber sido completamente definida como parte del Diseño Arquitectónico, y es deber del desarrollador asegurar que la misma este completa antes de comenzar la implementación. Para ciertos componentes, en especial si son componentes reutilizables o exponen servicios que son consumidos por terceros, la interfaz debe ser un contrato que cuyos cambios deben gestionarse con cuidado, y en ese caso un documento es ampliamente recomendado. En caso que las interfaces no tengan tal nivel de criticidad, se recomienda para su documentación el uso de herramientas de documentación automática, como *Javadoc*.

Asimismo, se cuentan entre los productos intermedios los ***Casos de Prueba*** que especifican a los conductores de pruebas, tanto para las pruebas unitarias como para las pruebas de integración. Un ***Caso de Pruebas*** consiste en una especificación de la verificación del comportamiento de un componente en un método o rutina en particular siguiendo un esquema de pruebas de caja negra: Se le provee una entrada para la cual se conoce previamente la salida, se compara la salida generada por el software contra la conocida y basado en esto se determina si el comportamiento es o no el correcto. Cuando se desarrollan conductores de pruebas, deben especificarse diversos casos para la misma rutina del mismo componente. Es importante conseguir el equilibrio apropiado acerca de la cantidad de casos de pruebas que deben agregarse, procurando que siempre que se agregue algún caso de pruebas exista un valor agregado que el mismo provea, de otra manera, se recomienda no agregar tal caso de pruebas por resultar superfluo.

Dos importantes productos de trabajo intermedio son el ***Programa Fuente*** y el ***Programa Objeto***. El ***Programa Fuente*** consiste en cada uno de los módulos de código fuente usados para la implementación de un componente, en tanto el ***Programa Objeto*** es el producto de compilar el ***Programa Fuente***. Los Componentes de Software son construidos a partir de estos programas, por lo que tienen un importante papel en la generación de salidas del Proceso de Implementación.

La ***Secuencia de Integración*** consiste en un análisis previo, cuando aplica, del orden estricto en que debe realizarse una integración de componentes. Esta secuencia es de mucha utilidad para la integración manual, y típicamente se maneja como un documento que debe ser cuidadosamente gestionado.

El ***Guión de Generación de versiones ejecutables*** (comúnmente conocido como *script*) es un programa desarrollado en un lenguaje *scripting* que permite la rápida compilación, construcción y empaquetado de las versiones ejecutables de los ***Componentes de Software*** y/o el ***Producto de Software*** completo. Este guión debe ser desarrollado por el equipo de desarrollo, y provee una manera estandarizada para la generación de las versiones ejecutables, además de que puede incluirse todo tipo de aspectos de configuración de la aplicación durante la ejecución del *script*.

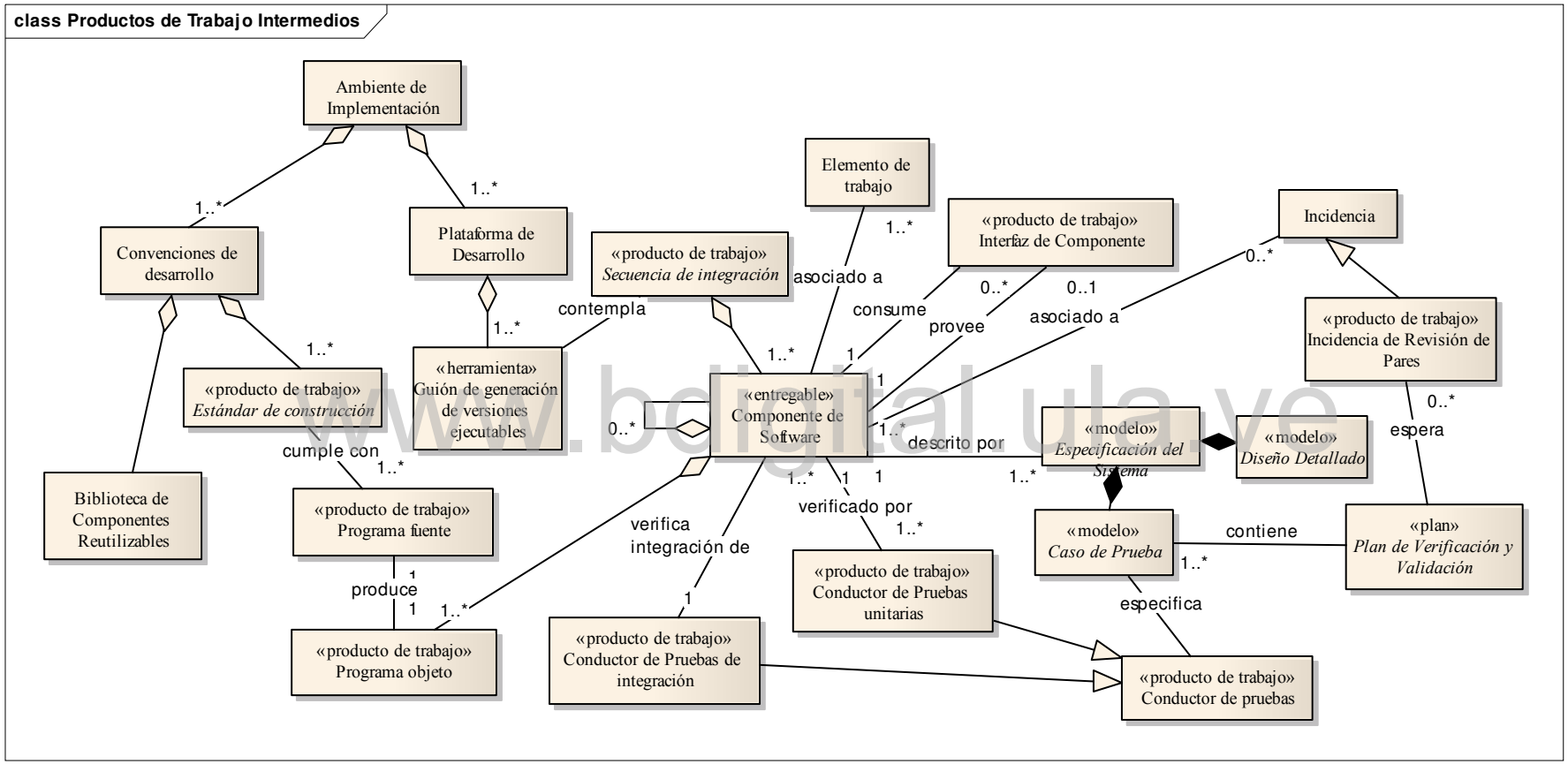


Figura 22. Diagrama de Clases de Productos de Trabajo Intermedios del Proceso de Implementación

Otro producto de trabajo intermedio que contempla nuestro modelo de productos es la ***Incidencia***. Ella se crea cuando se detecta una no conformidad durante la Revisión de Pares o existe algún aspecto del código que se sugiere sea cambiado. Tales incidencias tienen una estructura similar al ***Elemento de Trabajo***, por cuanto representa una tarea de ingeniería a realizar, no obstante sus representaciones no tienen por que ser exactas.

4 Herramientas de Soporte del Proceso de Implementación

Se consideran Herramientas de Soporte aquellos implementos automatizados que permiten soportar la ejecución de un Proceso de Implementación ágil y disciplinado. Aunque el Manifiesto Ágil (Highsmith y Fowler, 2001) es explícito al dar más valor a las personas y las interacciones que a los procesos y las herramientas, no es menos cierto que son diversas las prácticas del desarrollo ágil que serían imposibles de realizar de no contar con herramientas que automaticen tareas repetitivas que permitan acortar los ciclos de retroalimentación, como por ejemplo, Desarrollo Orientado a Pruebas (*Test-Driven Development*) (Ambler, 2003). En la figura 23, se encuentra el modelo de productos asociado a las herramientas que dan soporte al Proceso de Implementación propuesto.

El producto clave con el que comienza este diagrama es el de la ***Plataforma de Desarrollo***, que se expuso en secciones anteriores, y se presenta un caso similar con el ***Guión de generación de versiones ejecutables***.

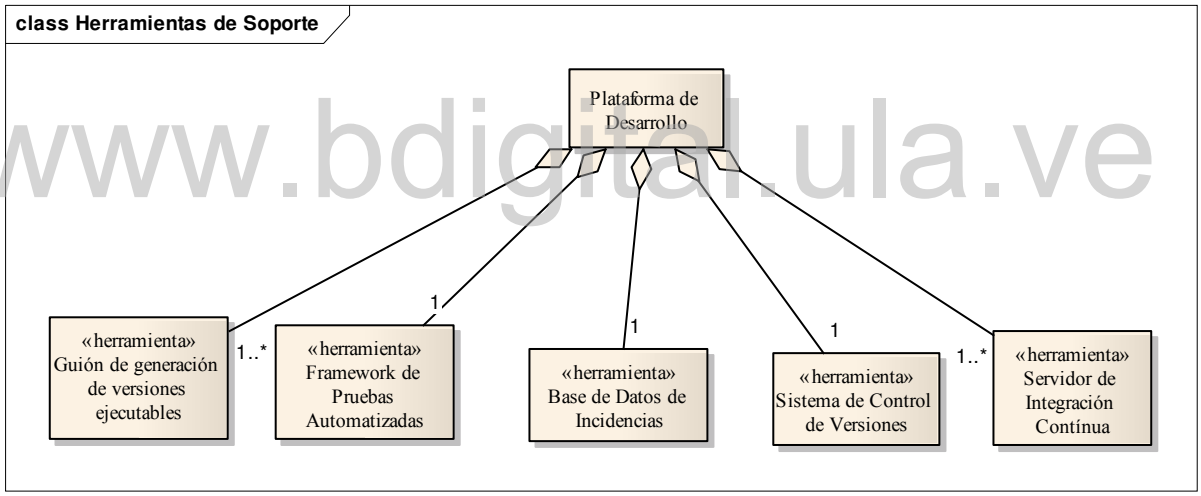


Figura 23. Diagrama de Clases de Herramientas de Soporte del Proceso de Implementación

Entre el resto de las herramientas se tienen las siguientes:

- ***Framework de Pruebas Automatizadas***: Consiste en un marco de trabajo que facilita enormemente la creación de conductores de pruebas. Existe un número nada despreciable de *framework*s como este completamente gratuitas y *open source* que proveen este tipo de funcionalidades.

- ***Base de Datos de Incidencias***: Una base de datos o un sistema de seguimiento de incidencias para la gestión de las mismas de mucha utilidad para los distintos procesos de validación y verificación. Las incidencias asociadas a la Revisión de Pares serían igualmente gestionadas con esta herramienta. Existen distintos sistemas con estas características también como software libre.

- ***Sistema de Control de Versiones***: Punto clave en el lograr la Integración Continua y la pertenencia colectiva del código. Un servicio de red que permite el manejo de las distintas versiones que puede tener algún componente.

- ***Servidor de Integración Continua***: Servicio que, integrado con el sistema de control de versiones, permite la constante integración y verificación de la misma sobre los componentes que haya.

5 Modelo de Productos Integrado

Habiendo mostrado previamente cada uno de los elementos considerados en el Modelo de Productos clasificados por su rol en el proceso (insumos, salidas e intermedios y herramientas), se procede ahora a mostrar el modelo integrado de estos conceptos, donde puede observarse la interacción entre estos elementos. El diagrama de clases mostrado en la figura 24 representa el modelo de productos del Proceso de Implementación ágil y disciplinado.

Se puede observar allí la relación entre las salidas expuestos (Ej., ***Componente de Software***) y los productos intermedios que son necesarios para su realización (ej. ***Programa Objeto*** y ***Programa Fuente***). De igual manera, se observa la relación existente entre las herramientas de soporte del proceso (por ej., el ***Framework de Pruebas Automatizadas***) y los productos de trabajo que se apoyan o se construyen haciendo uso de esas herramientas (para el ejemplo, los ***Conductores de Pruebas***).

6 Conclusiones de este capítulo

En este capítulo se describió detalladamente el modelo de productos que serán referenciados por los fragmentos situacionales de método que conforman Proceso de Implementación ágil y disciplinado que este trabajo propone. Tal modelo de productos se describió utilizando la notación de Diagramas de Clases de *UML*, que es uno de los estándares usados en la Ingeniería de Métodos Situacionales para tal fin.

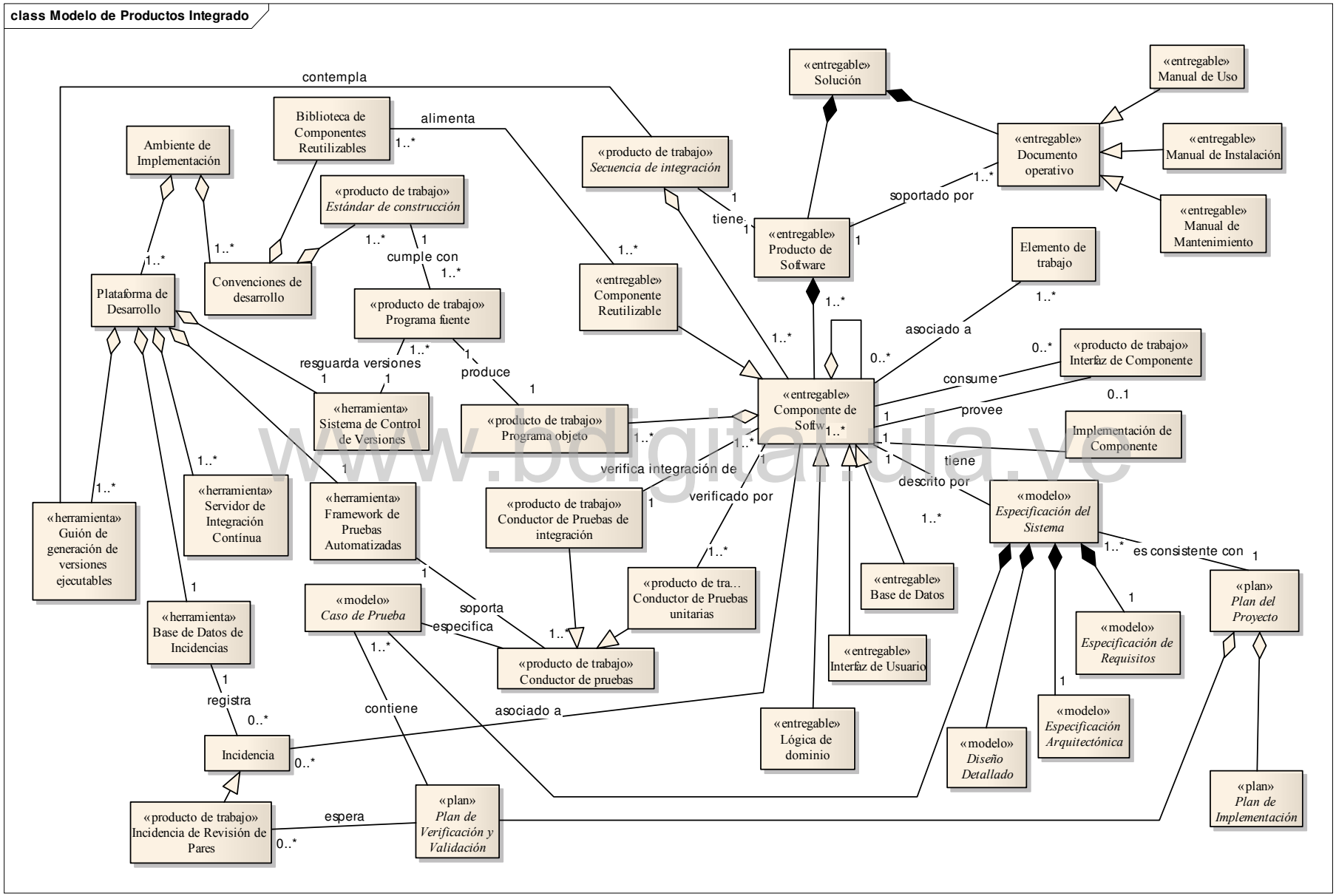


Figura 24. Diagrama de Clases del Modelo de Productos de Implementación de Software.

CAPITULO V: Modelo de Procesos de Implementación de Software

El presente capítulo contiene los fragmentos de proceso del método situacional para el Proceso ágil y disciplinado de Implementación de Software. Como se ha indicado en el capítulo III, el modelado se realiza siguiendo un enfoque orientado a decisión haciendo uso de las nociones de directivas y mapas de procesos orientados a decisión propuesta por Rolland, Prakash y Benjamen (Rolland et al, 1999) y la de Contextos descrita por Pohl, Dömges y Jarke (Pohl et al, 1994).

Dado que los elementos del enfoque orientado a decisión tienen como elementos esenciales en su definición a la intención y a la situación, se define como la intención del proceso macro de Implementación de Software la de Implementar los Elementos que pertenecen a una Solución, donde estos elementos son secciones parciales o totales de la Solución, del Producto de Software, uno o más Componentes de Software o uno o más Documentos operativos que soporten al Producto de Software. Del mismo modo, se define como la situación para el proceso macro el estado de la Solución o de los distintos elementos de la misma, y los elementos adicionales que se comentaron en la sección 4.1 del capítulo III.

1 Mapa Global: Implementar Elemento de Solución

Firma de la Directiva: `<(Solución); Implementar Elemento de Solución>`

La intención macro del Proceso de Implementación de software es implementar los elementos de la solución. Se hace uso de este nivel de granularidad para permitir la fácil interrelación con el resto de procesos en un método de características disciplinadas, y que pudiesen requerir la implementación de un componente de software a la vez, por lotes o toda la Solución de una sola vez. La figura 25 muestra el mapa global que satisfaría la intención macro del Proceso de Implementación.

La ejecución del Proceso de Implementación de Software puede tener como evento disparador alguno de los siguientes: 1) Implementación del primer Componente de Software de la Solución, lo que a su vez involucra la prueba final de preparación del Ambiente de Implementación; 2) Implementación de un nuevo Componente de Software adicional de la Solución; 3) Modificación de un componente existente ante un cambio o incidencia acontecida; 4) Ocurrencia de un evento de verificación técnica de la implementación de la Solución; y 5) Necesidad de creación o actualización de algún manual técnico requerido como parte de la Solución.

Este mapa está conformado por las siguientes intenciones: Proveer Componente, que busca hacer disponible el componente requerido ya sea por desarrollo o reutilización; Implantar Ambiente, que se plantea la meta de gestión de la plataforma y los estándares de implementación; Asegurar Calidad del Código, que involucra actividades de verificación del código fuente desarrollado para un componente o conjunto de componentes; Actualizar Documento Operativo, intención que apunta a la creación y actualización de los documentos técnicos operativos que se consideren como parte de la Solución; e Integrar Componente, que se encarga de orquestar el o los componentes que están siendo implementados con aquellos que han sido implementados en el pasado. La tabla 3 muestra los caminos de decisión disponibles en la intención global Implementar Elemento de Solución. Por su parte, las tablas 4, 5 y 6 muestran las directivas de selección de intención, selección de estrategia y ejecución de intención del mapa global, respectivamente.

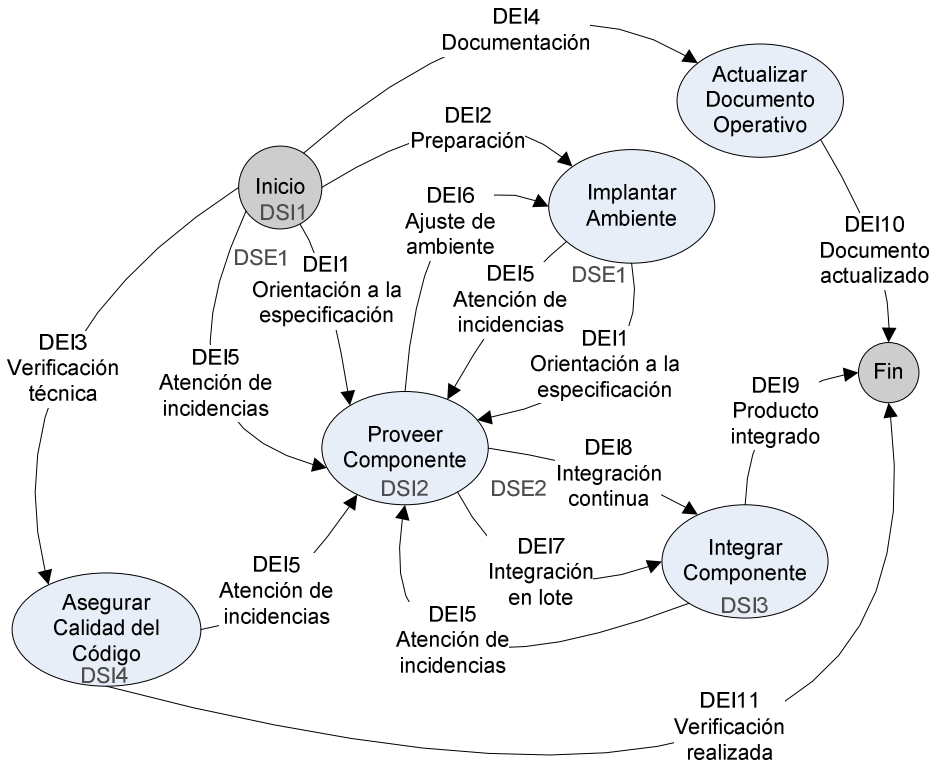


Figura 25. Mapa Global del Proceso de Implementación de Software.

Tabla 3. Tabla de caminos del mapa global de Implementación de Software

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI1	Proveer Componente	DSE1	Orientación a la especificación	DEI1
				Atención de incidencias	DEI5
		Implantar Ambiente	-	Preparación	DEI2
		Asegurar Calidad del Código	-	Verificación técnica	DEI3
		Actualizar Documento Operativo	-	Documentación	DEI4
Implantar Ambiente	-	Proveer Componente	DSE1	Orientación a la especificación	DEI1
				Atención de incidencias	DEI5
Proveer	DSI2	Implantar Ambiente	-	Ajuste de ambiente	DEI6

Componente		Integrar Componente	DSE2	Integración en lote	DEI7
				Integración continua	DEI8
Integrar Componente	DSI3	Proveer Componente	-	Atención de incidencias	DEI5
		Fin	-	Producto integrado	DEI9
Actualizar Documento Operativo	-	Fin	-	Documento actualizado	DEI10
Asegurar calidad del código	DSI4	Proveer Componente	-	Atención de incidencias	DEI5
		Fin	-	Verificación realizada	DEI11

Tabla 4. Tabla de directivas de selección de intención del Mapa Global de Implementación de Software

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI1	<(Solución); Avanzar desde Inicio>	DSE1: Seleccionar (<(Solución); Avanzar hacia Proveer Componente>) **U**	A1
		DEI2: Seleccionar (<(Especificación); Implantar Ambiente por la estrategia de preparación>) **U**	A2
		DEI3: Seleccionar (<(Plan de Verificación y Validación); Asegurar Calidad del Código por la estrategia de verificación técnica>) **U**	A3
		DEI4: Seleccionar (<(Especificación); Actualizar Documento Operativo por la estrategia de documentación>)	A4
DSI2	<(Solución); Avanzar desde Proveer Componente>	DEI6: Seleccionar (<(Especificación); Implantar Ambiente por la estrategia de ajuste de ambiente>) **U**	A5
		DSE2: Seleccionar (<(Solución); Avanzar hacia Integrar Componente>)	A6
DSI3	<(Solución); Avanzar desde Integrar Componente>	DEI9: Seleccionar (<(Solución); Finalizar con la estrategia de producto integrado>) **U**	A7
		DEI5: Seleccionar (<(Componente con estado (Componente) = Revisado); Proveer Componente por la estrategia de atención de incidencias>)	A8
DSI4	<(Solución); Avanzar desde Asegurar Calidad del Código>	DEI5: Seleccionar (<(Componente con estado (Componente) = Revisado); Proveer Componente por la estrategia de atención de incidencias>) **U**	A9
		DEI11: Seleccionar (<(Solución); Finalizar con la estrategia de verificación realizada>)	A10

Argumentos de Directivas

- **A1:** El *Ambiente de Implementación* se encuentra configurado para el proyecto actual, y se requiere la creación o modificación en la implementación de uno o más componentes de software.
- **A2:** El *Ambiente de Implementación* está instalado pero no se ha configurado apropiadamente en lo mínimo necesario para las actividades de implementación del elemento de solución requerido.
- **A3:** De acuerdo al *Plan de Verificación y Validación*, se requiere la ejecución de un proceso de revisión técnica
- **A4:** Se requiere la creación o actualización de un manual de soporte del *Producto de Software*.
- **A5:** El *Ambiente de Implementación* ya está configurado, pero en el reciente proceso de detallar un componente se observó la necesidad de cambio en el mencionado ambiente.
- **A6:** La especificación de los componentes aprovisionados hasta el momento representa el insumo suficiente y necesario para ejecutar el proceso de integración de acuerdo a la estrategia acordada para ello.
- **A7:** Todos los componentes de la especificación original para la ejecución actual del Proceso de Implementación se encuentran aprovisionados e integrados en la solución.
- **A8:** Durante la integración de componentes uno o más componentes requirieron revisión a nivel de sus interfaces o surgió algún problema durante la misma que requiere ser corregido por el equipo de desarrollo
- **A9:** Algunas de las incidencias detectadas son de fácil y rápida resolución.
- **A10:** No se consiguió incidencia alguna que deba resolverse o las encontradas son lo suficientemente impactantes como para merecer una planificación previa a la resolución de las mismas.

Tabla 5. Tabla de directivas de selección de estrategia del Mapa Global de Implementación de Software

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE1	<(Solución); Avanzar hacia Proveer Componente>	DEI1: Seleccionar(<(Solución); Proveer Componente por la estrategia de orientación a la especificación>) **U**	A11
		DEI2: Seleccionar(<(Solución); Proveer Componente por la estrategia de atención a incidencias>)	A12
DSE2	<(Solución); Avanzar hacia Integrar Componente>	DEI7: Seleccionar (<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la estrategia de integración en lote>) **U**	A13
		DEI8: Seleccionar(<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la estrategia de integración continua>)	A14

Argumentos de Directivas

- **A11:** El *Plan de Implementación* involucra la creación de nuevos componentes o la modificación de un componente existente por el mecanismo acordado para la gestión de cambios.

- **A12:** Se ejecuta el *Proceso de Implementació*n debido a alguna *Incidencia* detectada durante la fase de validación para realizar ajustes a uno o más *Componentes de Software*.

- **A13:** Los *Componentes de Software* son desarrollados por diversos equipos disjuntos con gestión separada, o las características inherentes de los componentes a integrar dificultan en gran medida la ejecución constante o automatizada del proceso de integración por parte de cada miembro del equipo de desarrollo, o no se tiene experiencia en prácticas de Integración Continua.

- **A14:** El equipo de desarrollo es gestionado de manera uniforme y las características inherentes de los componentes facilitan la automatización y ejecución constante del proceso de compilación e integración de los componentes. De igual manera, los miembros del equipo de desarrollo tienen experiencia en las prácticas asociadas a la integración continua.

Tabla *6*. Tabla de directivas de ejecución de intención del Mapa Global de Implementación de Software

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI1	<(Solución); Proveer Componente por la estrategia de orientación a especificación>	Ver Mapa Local 2.1.
DEI2	<(Especificación); Implantar Ambiente por la estrategia de preparación>	Ver Mapa Local 2.3
DEI3	<(Plan de Verificación y Validación); Asegurar calidad del código por la estrategia de verificación técnica>	Ver Mapa Local 2.8
DEI4	<(Especificación); Actualizar documento operativo por la estrategia de documentación>	Ver Mapa Local 2.7
DEI5	<(Solución); Proveer Componente por la estrategia de atención a incidencias>	Ver Mapa Local 2.2
DEI6	<(Especificación); Implantar Ambiente por la estrategia de ajuste de ambiente>	Ver Mapa Local 2.4
DEI7	<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la estrategia de integración en lote>	Ver Mapa Local 2.5
DEI8	<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la estrategia de integración continua>	Ver Mapa Local 2.6
DEI9	<(Producto de Software, con estado(Producto de Software) = Integrado); Finalizar por la estrategia de producto integrado>	Ver Contexto 1.1
DEI10	<(Documento Operativo, con estado(Documento Operativo) = Actualizado); Finalizar por la estrategia de documento actualizado>	Ver Contexto 1.2
DEI11	<(Solución); Finalizar por la estrategia de verificación realizada>	Ver Contexto 1.3

1.1 Contexto: Finalizar por la estrategia de producto integrado

Tabla *7*. Definición del contexto Finalizar por la estrategia de producto integrado

<(Producto de Software, con estado(Producto de Software) = Integrado); Finalizar por la estrategia de producto integrado>
Definición del contexto: El proceso de integrar los componentes puede darse por satisfecho cuando los distintos componentes aprovisionados se pueden verificar tanto por separado como integradamente, y las pruebas realizadas no arrojan incidencia o error alguno. En este caso, el *Producto de Software* se considera integrado, y de esa manera el Proceso de Implementación culmina logrando sus objetivos.

1.2 Contexto: Finalizar por la estrategia de documento actualizado

Tabla *8*. Definición del contexto Finalizar por la estrategia de documento actualizado

<(Documento Operativo, con estado(Documento Operativo) = Actualizado); Finalizar por la estrategia de documento actualizado>
Definición del contexto: Una vez que el Documento Operativo ha sido actualizado y es consistente con las especificaciones indicadas y la versión del componente que se desea soportar, el Proceso de Implementación termina..

1.3 Contexto: Finalizar por la estrategia de verificación realizada

Tabla 9. Definición del contexto Finalizar por la estrategia de verificación realizada

<(Solución); Finalizar por la estrategia de verificación realizada>
Definición del contexto: Al haberse desarrollado las actividades de Verificación Técnica y documentado y notificado las incidencias que puedan haber sido descubiertas, el proceso se da por terminado.

2 Mapas Locales

2.1 Proveer Componente por la estrategia de orientación a la especificación

Firma de la Directiva: `<(Solución); Proveer Componente por la estrategia de orientación a la especificación>`

Este mapa viene a satisfacer una de las principales metas del Proceso de Implementación como lo es el aprovisionamiento de los componentes. Esta actividad busca hacer disponible un *Componente de Software* de la *Solución* requerido por la vía de desarrollo del mismo o bien por reutilización de un componente encontrado en la *Biblioteca de Componentes Reutilizables*. La estrategia de orientación a la especificación busca el cumplimiento de las responsabilidades asignadas al componente a implementar tal como se indica en la especificación de la Solución, ya sea durante la implementación inicial de dicho componente o ante un cambio ocurrido en la especificación fuera del ámbito del Proceso de Implementación. Para las correcciones a un componente debido a no conformidades encontradas durante las actividades de Verificación y Validación (V&V),

se debe usar la estrategia de atención de incidencias (Mapa Local 2.2). La figura 26 muestra el mapa local de Proveer Componente por la estrategia de orientación a la especificación.

Este mapa de procesos incluye las siguientes intenciones: Detallar Componente, que busca establecer la lista de actividades necesarias para el aprovisionamiento del componente a partir de su especificación. También busca determinar si las responsabilidades del *Componente de Software* pueden ser satisfechas mediante la reutilización de un componente disponible en la *Biblioteca de Componentes Reutilizables*, o si por el contrario tenga que ser codificado; Reutilizar Componente, que aprovecha los componentes reutilizables que se encuentran en las *Bibliotecas de Componentes Reutilizables* establecidas para satisfacer las necesidades de componentes en la aplicación, cuando así pueda hacerse; Codificar Componente, que permite el aprovisionamiento de componentes mediante la codificación parcial o total de las funcionalidades necesarias, ejecutando así la actividades que fueron racionalizadas por el desarrollador durante el proceso del detallado del diseño del componente; Depurar Componente, que permite ubicar defectos o errores en la codificación de un componente para así corregirlos posteriormente; y Verificar Componente, que busca certificar que el componente provisto cumpla con las especificaciones dadas, usando para esto las actividades de pruebas unitarias de componentes con sus respectivos conductores de pruebas.

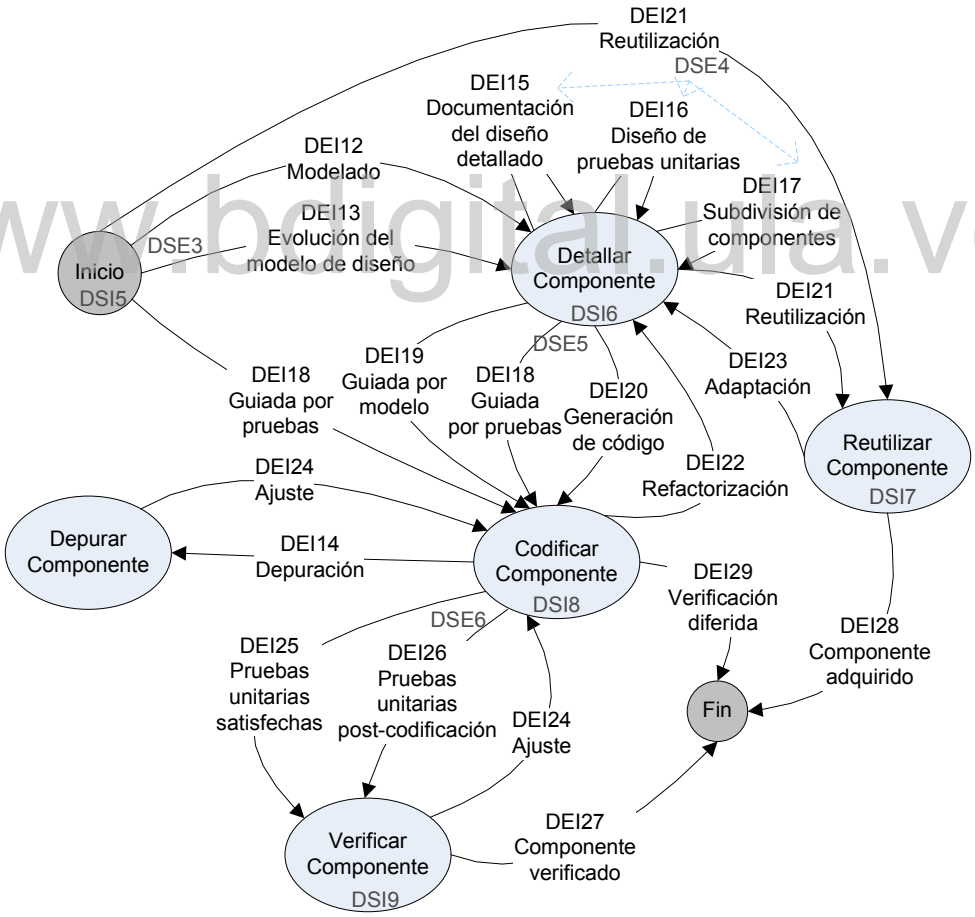


Figura 26. Mapa Local: Proveer Componente por la estrategia de orientación a la especificación

La tabla 10 muestra los caminos entre las intenciones y las estrategias que pueden ser utilizadas para lograr tales intenciones y permite observar las referencias con las directivas de selección de intención, selección de

estrategia y ejecución de intención. De manera seguida se presentan las listas de las directivas mencionadas anteriormente, en las tablas 11 (Directivas de Selección de Intención), 12 (Directivas de Selección de Estrategia) y 13 (Directivas de Ejecución de intención).

Tabla 10. Tabla de caminos del mapa local de Proveer Componente por la estrategia de orientación a la especificación

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección, de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI5	Detallar Componente	DSE3	Modelado	DEI12
				Evolución del modelo de diseño	DEI13
		Reutilizar Componente	-	Reutilización	DEI21
		Codificar Componente	-	Guiada por pruebas	DEI18
Detallar Componente	DSI6	Detallar Componente	DSE4	Documentación del diseño detallado	DEI15
				Diseño de pruebas unitarias	DEI16
				Subdivisión de componentes	DEI17
		Reutilizar Componente	-	Reutilización	DEI21
		Codificar Componente	DSE5	Guiada por pruebas	DEI18
				Guiada por modelo	DEI19
				Generación de código	DEI20
Reutilizar Componente	DSI7	Detallar Componente	-	Adaptación	DEI23
		Fin	-	Componente adquirido	DEI28
Codificar Componente	DSI8	Detallar Componente	-	Refactorización	DEI22
		Depurar Componente	-	Depuración	DEI14
		Verificar Componente	DSE6	Pruebas unitarias satisfechas	DEI25
				Pruebas unitarias post-codificación	DEI26
		Fin	-	Verificación diferida	DEI29

Depurar Componente	-	Codificar Componente	-	Ajuste	DEI24
Verificar Componente	DSI9	Codificar Componente	-	Ajuste	DEI24
		Fin	-	Componente verificado	DEI27

Tabla 11. Tabla de directivas de selección de intención de Proveer Componente por la estrategia de orientación a especificación

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI5	<(Especificación); Avanzar desde Inicio>	DSE3: Seleccionar(<(Especificación); Avanzar hacia Detallar Componente >) **U**	A15
		DEI18: Seleccionar (<(Especificación); Codificar Componente por la estrategia guiada por pruebas>) **U**	A16
		DEI21: Seleccionar (<(Especificación); Reutilizar Componente por la estrategia de reutilización>)	A17
DSI6	<(Especificación);Avanzar desde Detallar Componente>	DSE4: Seleccionar(<(Modelo Detallado); Avanzar hacia Detallar Componente>) **U**	A18
		DEI21: Seleccionar(<(Diseño Detallado); Reutilizar Componente por la estrategia de reutilización>) **U**	A19
		DSE5: Seleccionar(<(Especificación); Avanzar hacia Codificar Componente>)	A20
DSI7	<(Solución);Avanzar desde Reutilizar Componente>	DEI23: Seleccionar (<(Diseño Detallado); Detallar Componente por la estrategia de adaptación>) **U**	A21
		DEI28: Seleccionar (<(Solución); Finalizar por la estrategia de componente adquirido>)	A22
DSI8	<(Componente);Avanzar desde Codificar Componente>	DEI22: Seleccionar(<(Programa Fuente);Detallar Componente por la estrategia de refactorización>) **U**	A23
		DEI14: Seleccionar(<(Componente, con estado (Componente) = Ejecutable); Depurar Componente por la estrategia de depuración>) **U**	A24
		DSE6: Seleccionar(<(Solución); Avanzar hacia Verificar Componente>) **U**	A25
		DEI29: Seleccionar(<(Solución); Finalizar por la estrategia de verificación diferida>)	A26
DSI9	<(Componente); Avanzar desde Verificar Componente>	DEI24: Seleccionar(<(Componente, con estado(Componente) = Requiere Ajuste); Codificar Componente por la estrategia de ajuste>) **U**	A27

		DEI27: Seleccionar(<(Componente, con estado(Componente) = Verificado); Finalizar por la estrategia de componente verificado>)	A28

Argumentos de Directivas

- **A15:** La *Especificación* aun tiene un nivel alto y necesita ser más concreta antes de comenzar la codificación.

- **A16:** El entendimiento que el desarrollador tiene del componente a desarrollar es muy completo, la especificación se puede codificar directamente y ya se tienen bien conceptualizadas los casos de pruebas unitarias que deben ser diseñadas. De igual manera, el desarrollador entiende los conceptos asociados al desarrollo orientado a pruebas.

- **A17:** Se observa en la especificación del componente que el mismo puede ser provisto a partir de la reutilización de un componente disponible en la biblioteca de componentes reutilizables del proyecto o bien puede ser adquirido como un COTS.

- **A18:** Aún se detectan interrogantes de implementación en el modelo detallado que deben ser respondidas antes de comenzar la codificación, por lo que se requiere un mayor nivel de granularidad en el detalle del mencionado modelo o faltan actividades que deben ser realizadas como parte del diseño detallado (por ejemplo, diseño de pruebas unitarias o la documentación del diseño detallado).

- **A19:** Se ha detectado que la funcionalidad requerida para el componente que se está detallando puede ser cubierta de manera parcial o total por un componente disponible en alguna de las bibliotecas de componentes reutilizables del proyecto.

- **A20:** El diseño detallado está suficientemente elaborado, las especificaciones de pruebas unitarias requeridas han sido generadas, se entienden las actividades de codificación que deben ser realizadas y se han cubierto las expectativas de documentación que se requiere durante el diseño detallado del componente.

- **A21:** El componente reutilizable requiere una adaptación antes de que pueda ser utilizado en la solución en desarrollo.

- **A22:** El componente reutilizable puede ser reutilizado sin cambio alguno, y el mismo ya está disponible en los repositorios de componentes del proyecto.

- **A23:** Durante la codificación se detectó que el diseño puede ser implementado de manera más simplificada, más legible, que pueda evitar duplicación de código o que con el ajuste del diseño del código existente se reduce la cantidad de trabajo de implementación del componente actual.

- **A24:** Se ha detectado un comportamiento no esperado durante las pruebas iniciales del componente y no es evidente la razón del mismo.

- **A25:** Todos los elementos de trabajo asociados a la codificación han sido satisfechos y el componente ha aprobado las pruebas preliminares durante la codificación.

- **A26:** Se ha completado la codificación sin pruebas unitarias de un componente que en su especificación o en los estándares se considera muy difícil de verificar de manera unitaria.
- **A27:** Durante la verificación se ha detectado una anomalía en el componente que debe ser corregida.
- **A28:** El componente ha pasado satisfactoriamente las pruebas unitarias y cumple con los requisitos de interfaz que dicta la especificación.

Tabla 12. Tabla de directivas de selección de estrategias de Proveer Componente por la estrategia de orientación a especificación

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE3	<(Especificación);Avanzar hacia Detallar Componente>	DEI12: Seleccionar(<(Especificación); Detallar Componente por la estrategia de modelado>) **U**	A29
		DEI13: Seleccionar(<(Especificación); Detallar Componente por la estrategia de evolución del modelo de diseño>)	A30
DSE4	<(Modelo Detallado); Avanzar hacia Detallar Componente>	DEI15: Seleccionar(<(Modelo Detallado); Detallar Componente por la estrategia de documentación del diseño detallado>) **U**	A31
		DEI16: Seleccionar(<(Modelo Detallado); Detallar Componente por la estrategia de diseño de pruebas unitarias>) **U**	A32
		DEI17: Seleccionar(<(Modelo Detallado); Detallar Componente por la estrategia de subdivisión de componentes>) **U**	A33
DSE5	<(Especificación); Avanzar hacia Codificar Componente>	DEI18: Seleccionar(<(Especificación); Codificar Componente por la estrategia guiada por pruebas>) **U**	A34
		DEI19: Seleccionar(<(Diseño Detallado); Codificar Componente por la estrategia guiada por modelo>) **U**	A35
		DEI20: Seleccionar(<(Diseño Detallado); Codificar Componente por la estrategia de generación de código>)	A36
DSE6	<(Solución); Avanzar hacia Verificar Componente>	DEI25: Seleccionar(<(Componente), Verificar Componente por la estrategia de pruebas unitarias satisfechas>) **U**	A37
		DEI26: Seleccionar(<(Componente); Verificar Componente por la estrategia de pruebas unitarias post-codificación>)	A38

Argumentos de Directivas

- **A29:** La especificación se encuentra disponible en un formato que dificulta la generación automática del esqueleto del código o no se desea hacer uso de este tipo de facilidad.

- **A30:** La especificación del componente se encuentra disponible en un formato digital, haciendo uso de una herramienta que provee la capacidad de generación automática de esqueleto de código y se considera que aprovechar tal funcionalidad es eficiente para la codificación del componente.

- **A31:** El documento de diseño detallado es un requisito explícito del proyecto, ya sea por solicitud del cliente o por que haya sido acordado de esa manera, por estándares organizacionales. Además, la complejidad de la implementación merece una explicación más detallada que la disponible en la especificación arquitectónica. Finalmente, se requiere que el diseño esté suficientemente elaborado como para que no se invierta tiempo innecesario documentando algo que tiene alta tendencia a ser cambiado, por no haber sido comprobado aún.

- **A32:** El diseño detallado ya contempla la separación de las responsabilidades entre clases, se ha completado el diseño de las interfaces del componente y se desea hacer uso del desarrollo guiado por pruebas.

- **A33:** Se ha detectado durante el modelado la necesidad de separar o delegar en clases distintas partes de las responsabilidades que la arquitectura atribuye al componente especificado. También es útil cuando se detectan oportunidades de reutilización de una funcionalidad o características inherentes a una clase, para su aprovechamiento dentro o fuera del proyecto.

- **A34:** Se desea seguir la práctica de desarrollo guiado por pruebas, se posee el diseño preliminar de las pruebas unitarias a implementar, se cuenta con un *Framework* para el desarrollo de *Conductores Automatizados de Pruebas Unitarias* para el componente en desarrollo y el desarrollador tiene contexto básico del desarrollo de pruebas unitarias y sus conductores.

- **A35:** No se dispone de alguno de los requisitos del desarrollo guiado por pruebas o con la capacidad de generar código a partir del diseño detallado digitalizado del componente.

- **A36:** El diseño detallado se encuentra en un formato que puede ser usado por una herramienta capaz de generar automáticamente la estructura base del código.

- **A37:** Se ha seguido la práctica de desarrollo guiado por pruebas y las pruebas unitarias se ejecutan de manera correcta en su totalidad.

- **A38:** No se dispone de pruebas unitarias para el componente actual, sin embargo ya las actividades de codificación se concluyeron.

Tabla 13. Tabla de directivas de selección de estrategias de Proveer Componente por la estrategia de orientación a especificación

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI12	`<(Especificación); Detallar Componente por la estrategia de modelado>`	Ver Contexto 2.1.1
DEI13	`<(Especificación); Detallar Componente por la estrategia de evolución del modelo de diseño>`	Ver Contexto 2.1.2
DEI14	`<(Componente, con estado (Componente) = Ejecutable); Depurar Componente por la estrategia de depuración>`	Ver Contexto 2.1.13

DEI15	<(Modelo Detallado); Detallar Componente por la estrategia de documentación del diseño detallado>	Ver Contexto 2.1.3
DEI16	<(Modelo Detallado); Detallar Componente por la estrategia de diseño de pruebas unitarias>	Ver Contexto 2.1.4
DEI17	<(Especificación); Detallar Componente por la estrategia de subdivisión de componentes>	Ver Contexto 2.1.5
DEI18	<(Especificación); Codificar Componente por la estrategia guiada por pruebas>	Ver Contexto 2.1.8
DEI19	<(Diseño Detallado); Codificar Componente por la estrategia guiada por modelo>	Ver Contexto 2.1.9
DEI20	<(Modelo Detallado>; Codificar Componente por la estrategia de generación de código>	Ver Contexto 2.1.10
DEI21	<(Especificación);Reutilizar Componente por la estrategia de reutilización>	Ver Contexto 2.1.12
DEI22	<(Programa Fuente); Detallar Componente por la estrategia de refactorización>	Ver Contexto 2.1.6
DEI23	<(Especificación, Interfaz de componente reusable); Detallar Componente por la estrategia de adaptación>	Ver Contexto 2.1.7
DEI24	<(Componente, con estado(Componente) = Requiere Ajuste); Codificar Componente por la estrategia de ajuste>	Ver Contexto 2.1.11
DEI25	<(Componente), Verificar Componente por la estrategia de pruebas unitarias satisfechas>	Ver Contexto 2.1.14
DEI26	<(Componente); Verificar Componente por la estrategia de pruebas unitarias post-codificación>	Ver Contexto 2.1.15
DEI27	<(Componente, con estado(Componente) = Verificado); Finalizar por la estrategia de componente verificado>	Ver Contexto 2.1.16
DEI28	<(Solución); Finalizar por la estrategia de componente adquirido>	Ver Contexto 2.1.17
DEI29	(<(Solución); Finalizar por la estrategia de verificación diferida>	Ver Contexto 2.1.18

2.1.1 Contexto: Detallar Componente por la estrategia de modelado

El proceso de detallar el componente consiste en convertir las especificaciones funcionales y arquitectónicas en actividades de codificación fácilmente realizables e inteligibles para cualquier desarrollador. Esta es una actividad mental que no necesariamente sigue una secuencia explícita de pasos; no obstante, puede ser descrita como el proceso de extracción, a partir de la especificación del componente, de una lista de elementos funcionales o no funcionales que pueden verificarse como parte del componente una vez que se ha implementado. Tales elementos pudiesen ser:

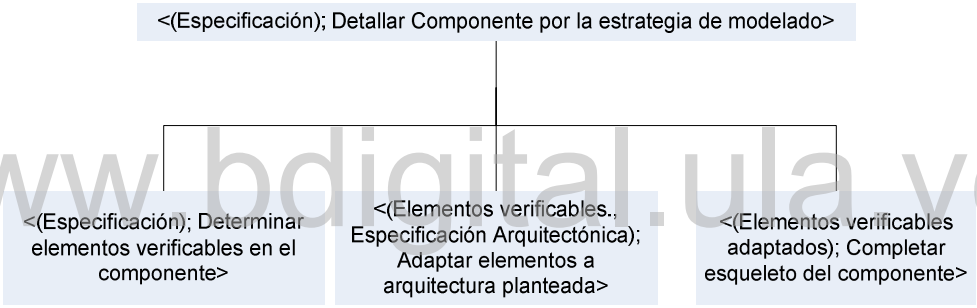
- Aspectos visibles de la funcionalidad (elementos de Interfaz de Usuario (UI por sus siglas en inglés), opciones explícitas de la interfaz de usuario),
- Características de la interfaz de uso del componente,

- Características, en su forma o contenido, de la salida resultante del componente o
- La definición de una expectativa de comportamiento interno del componente.

Estos elementos deben ser adaptados a la arquitectura previamente definida para el tipo de componente que se modela y que forma parte de la especificación aportada para el desarrollo del mismo.

La estructura del componente puede o no haber sido detallada como parte de la especificación inicial, dependiendo del alto o bajo nivel que tenga la responsabilidad asignada al mencionado componente. En cualquiera de estas situaciones, se espera que la estructura sea completada durante el proceso de detallado del componente, refiriéndose esto a la definición específica de las firmas de métodos, funciones o procedimientos a ser desarrollados. La tabla 14 muestra la definición de este contexto

Tabla 14. Definición del Detallar Componente por la estrategia de modelado

<(Especificación); Detallar Componente por la estrategia de modelado>
Definición del contexto: 

2.1.2 Contexto: Detallar Componente por la estrategia de evolución del modelo de diseño

En aquellos casos en que la especificación se encuentra disponible en un formato digital que puede ser consumido por una herramienta que facilite la generación de código puede aplicarse el contexto aquí definido.

El proceso consiste en la depuración sucesiva del modelo en el formato digital, considerando la especificación funcional y arquitectónica también allí disponible, con el fin de asegurar que la estructura generada contempla una definición completa de la estructura del componente.

En estos casos lo más común consiste en comenzar por un proceso inicial de preparación del modelo disponible para la generación de código; luego se procede con una generación del código y la revisión del resultado para verificar que el código resultante sea el esperado. En el caso en que el resultado no satisfaga las expectativas, se procede con la depuración del digital para repetir el ciclo a partir de una generación del código. La tabla 15 en la página siguiente muestra la definición de este contexto.

2.1.3 Contexto: Detallar Componente por la estrategia de documentación del diseño detallado

Una vez disponible el modelo detallado del componente y cuando se requiera la documentación explícita del diseño detallado del componente, se puede generar el Documento de Diseño Detallado. Esta actividad de

documentación consiste en explicar la estructura y las estrategias o comportamientos que serán representados por la codificación del componente. La tabla 16 que encontrará más adelante contiene la definición de este contexto plan.

Tabla 15. Definición del contexto Finalizar Detallar Componente por la estrategia de evolución del modelo de diseño

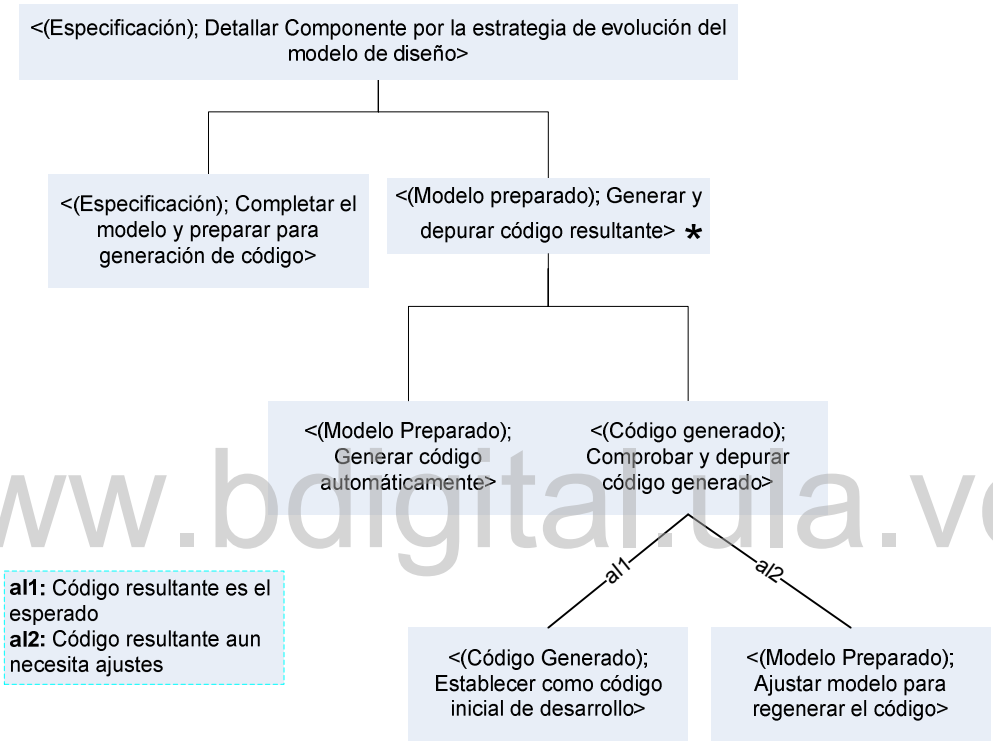
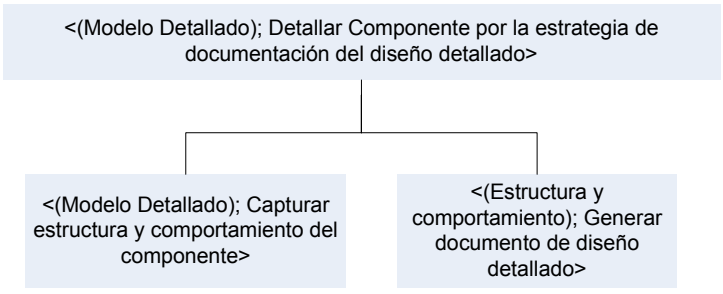
<(Especificación); Detallar Componente por la estrategia de evolución del modelo de diseño>
Definición del contexto: 

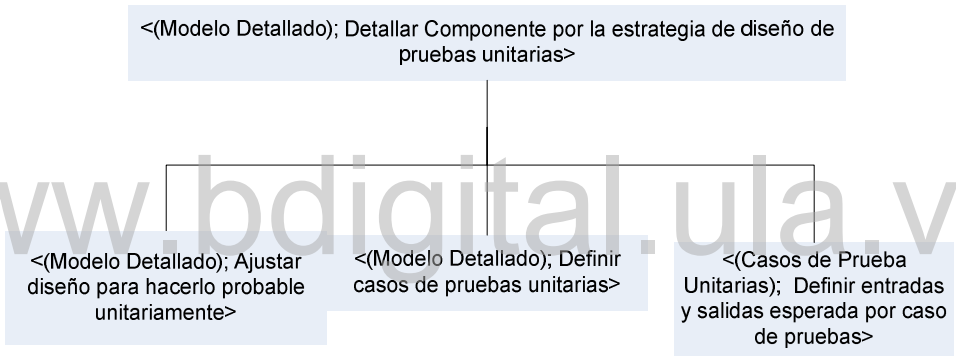
Tabla 16. Definición del contexto Detallar Componente por la estrategia de documentación del diseño detallado

<(Modelo Detallado); Detallar Componente por la estrategia de documentación del diseño detallado>
Definición del contexto: 

2.1.4 Contexto: Detallar Componente por la estrategia de diseño de pruebas unitarias

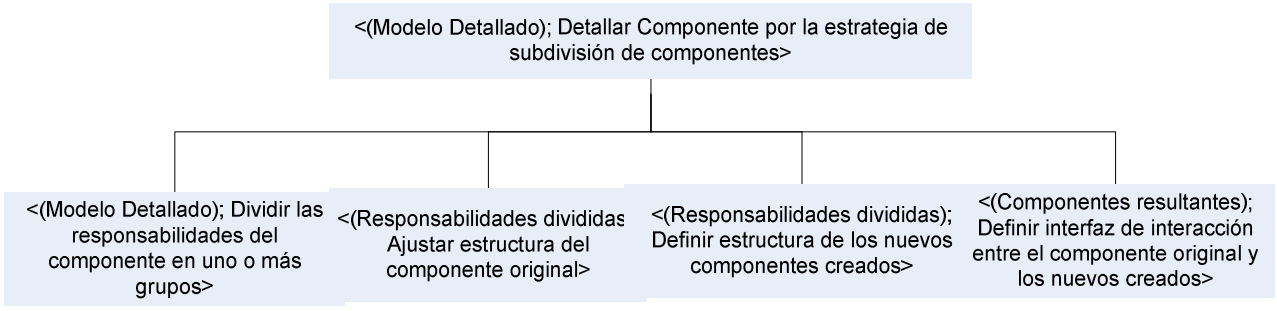
En el caso en que se desea hacer uso del Desarrollo Guiado por Pruebas, el diseño de los conductores automatizados de pruebas unitarias se convierte en parte del diseño detallado de los componentes. El proceso de diseño de estas pruebas unitarias representa primero asegurar la capacidad del componente de ser probado unitariamente, lo que representa la necesidad de poder aislarlo de sus dependencias de otros componentes al momento de la prueba. Esto es conocido como *Design For Testability (DFT)* (Payne et al, 1997). Una vez asegurada esta capacidad del componente se procede a la extracción de los casos de pruebas unitarias que consiste en la definición de casos de pruebas típicamente de caja blanca, basado en la dinámica de definición de parámetros de entrada para el componente y establecimiento de las expectativas de salida de acuerdo a la especificación como al modelo detallado previamente construido.

Tabla 17. Definición del contexto Detallar Componente por la estrategia de diseño de pruebas unitarias

<(Modelo Detallado); Detallar Componente por la estrategia de diseño de pruebas unitarias>
Definición del contexto: 

2.1.5 Contexto: Detallar Componente por la estrategia de subdivisión de componentes

Tabla 18. Definición del contexto Detallar Componente por la estrategia de subdivisión de componentes

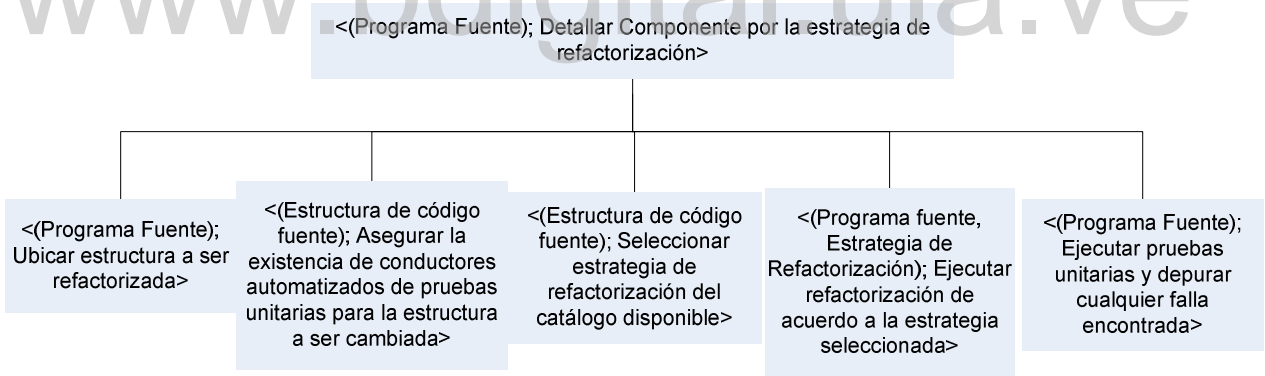
<(Modelo Detallado); Detallar Componente por la estrategia de subdivisión de componentes>
Definición del contexto: 

Ante la común situación de encontrarse con un componente que es candidato a ser subdividido en dos o más componentes para mantener la simplicidad del componente, se comienza por el proceso de la división de las responsabilidades originales del componente en dos o más grupos que permitan la asignación de responsabilidades simplificadas a cada componente, manteniendo la cohesión de cada nuevo componente. La estructura del componente original debe ser modificada para adaptarse a la existencia de nuevos componentes sobre los que delegará funcionalidades que originalmente estaban asignadas a el, y de igual manera debe definirse la estructura de el o los nuevos componentes creados. Finalmente, debe diseñarse la interfaz de interacción de los componentes creados con el componente original. La tabla 18 presenta la definición del contexto plan asociado a esta actividad.

2.1.6 Contexto: Detallar Componente por la estrategia de refactorización

La refactorización es el proceso de cambiar un sistema de software de manera que no se modifique el comportamiento externo del mismo pero si se mejore su estructura interna (Fowler et al, 2000). Consiste en el ajuste de la estructura de un componente para simplificar el código resultante, hacerlo más entendible y mantenible, sin con ello cambiar en forma alguna la funcionalidad o comportamiento del mencionado componente.

Tabla 19. Definición del contexto Detallar Componente por la estrategia de refactorización

<(Programa Fuente);Detallar Componente por la estrategia de refactorización>
Definición del contexto: 

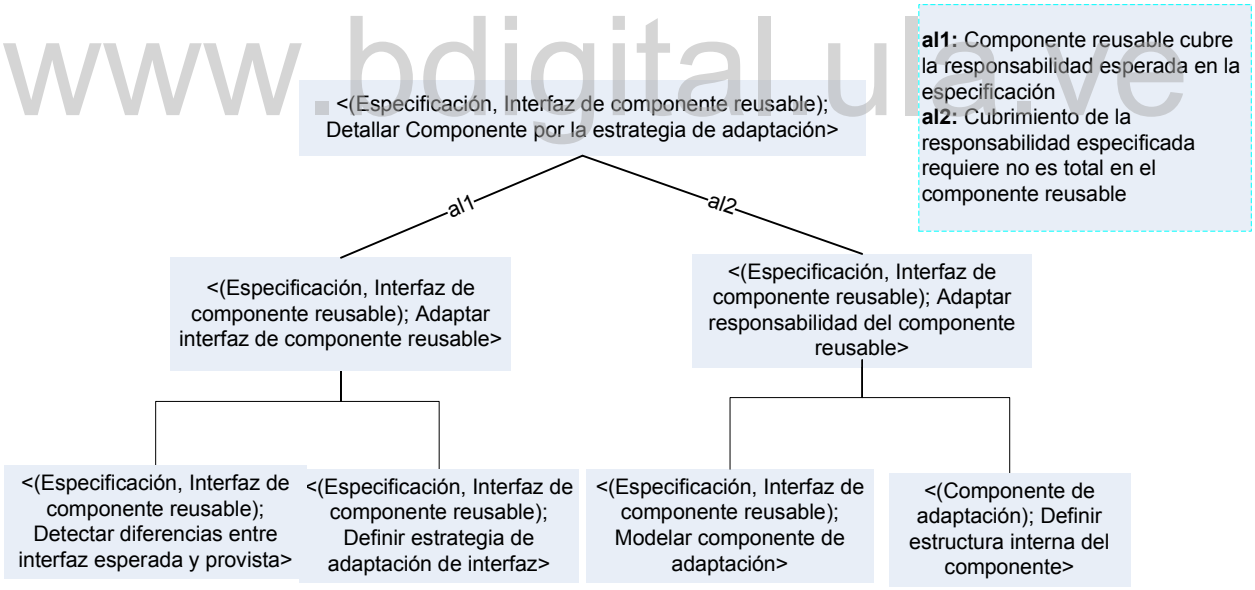
El proceso de refactorización, una vez que se ha determinado que es necesario realizarlo, comienza por la determinación de la estructura a ser cambiada. Se debe verificar que se tienen disponibles conductores automatizados de pruebas unitarias para las estructuras mencionadas, de no tenerlos deben ser construidos antes de continuar. Como paso siguiente se tiene determinación de la estrategia de refactorización apropiada, que pudiese ser seleccionada de un catálogo de estrategias como las presentadas en (Fowler et al, 2000), luego se realiza la ejecución del cambio sugerido por la estrategia seleccionada, lo cual debe ser corroborado con la ejecución de las pruebas unitarias que certifican la correcta realización del proceso. La tabla 19 presenta la definición de este contexto.

2.1.7 Contexto: Detallar Componente por la estrategia de adaptación

Una vez que se ha determinado el componente apropiado para ser reutilizado en el aprovisionamiento de un componente pero se observa que se requiere una adaptación en el componente a reutilizar, se debe crear el modelo detallado de dicha adaptación. Se inicia por la verificación de las diferencias detectadas entre la responsabilidad que debe tener el componente en el sistema en desarrollo y la responsabilidad para la que fue diseñado el componente reusable. En el caso de que el componente reusable cubra la misma responsabilidad pero la interfaz no es la misma, se procede a la adaptación de la interfaz, que consiste en detectar las diferencias entre las interfaces esperada y provista y en definir la estrategia de adaptación de la interfaz. Si por el contrario, el *Componente Reusable* sólo cubre una parte de la responsabilidad requerida, se procede a la creación de un envoltorio o *wrapper* que delegue en él la responsabilidad que sí cubre y se modela detalladamente lo necesario para complementarlo para que satisfaga las expectativas establecidas en la especificación. La tabla 20 muestra el contexto plan y los casos asociados a la actividad de diseño de la adaptación de componentes reusables.

Tabla 20. Definición del contexto Detallar Componente por la estrategia de adaptación

<(Especificación, Interfaz de componente reusable);Detallar Componente por la estrategia de adaptación>
Definición del contexto:

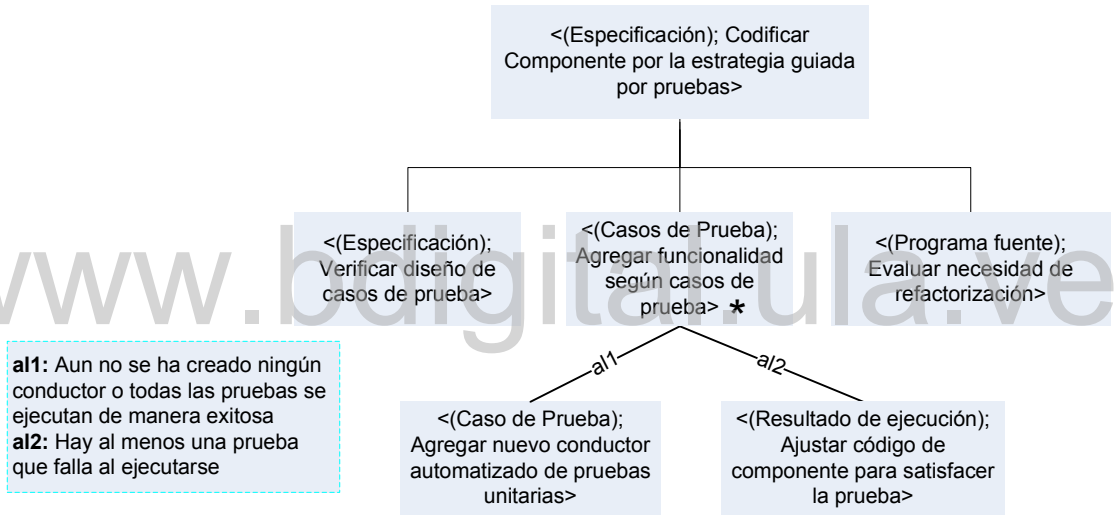


2.1.8 Contexto: Codificar Componente por la estrategia guiada por pruebas

El proceso de codificación de componentes siguiendo la estrategia guiada por pruebas responde al concepto conocido como *Test Driven Development*, práctica que ha sido popularizada por XP (Beck, 2000). Esta práctica consiste en la creación de las pruebas unitarias antes de la codificación del componente, por lo que la actividad de codificación se traduce en la búsqueda de código que satisfaga los casos de prueba previamente diseñados e implementados. El proceso comienza por agregar un conductor automatizado para un caso de prueba unitaria asociado al comportamiento que quiere codificarse en el componente. Dicho conductor es creado haciendo uso

de un *Framework* de Pruebas Unitarias Automatizadas, como se comentó en el modelo de productos del proceso, así como también se hace uso de dobles de prueba para los componentes de los que depende el objeto actual de pruebas. Una vez implementado el conductor se procede a su ejecución lo cual se espera arroje un resultado fallido de la ejecución, por lo que se procede a la creación o ajuste del código del componente para que satisfaga la prueba creada. Una vez lograda esta satisfacción, se procede a crear un nuevo conductor, hasta que todos los casos de pruebas hayan sido implementados y satisfechos. Como paso final, se requiere la evaluación del código del componente para determinar si se requiere refactorización del diseño del mismo. La tabla 21 que se encuentra más adelante muestra la definición del contexto de esta actividad.

Tabla 21. Definición del contexto Codificar Componente por la estrategia guiada por pruebas

<(Especificación); Codificar Componente por la estrategia guiada por pruebas>
Definición del contexto: 

2.1.9 Contexto: Codificar Componente por la estrategia guiada por modelo

La codificación de componente guiada por modelo se refiere a la manera tradicional de codificar, siguiendo el modelo que resultó del diseño detallado del componente. Típicamente es una actividad artesanal y como tal no sigue necesariamente una secuencia explícita de pasos. Una forma de describir esta actividad sería decir que consiste en seguir las actividades lógicas planeadas durante el modelado del componente, traduciendo cada elemento considerado en el modelo detallado a un elemento en el código del componente, y que concluiría por la codificación en el lenguaje de programación seleccionado para cada elemento. La tabla 22 muestra el contexto asociado a esta actividad.

2.1.10 Contexto: Codificar Componente por la estrategia de generación de código

Esta estrategia de codificación de componentes es factible si el modelo de diseño ha sido realizado usando una herramienta *CASE* y posee a su vez la capacidad de generación de código a partir del diseño digitalizado. Este tipo de herramientas permiten generar la estructura base o esqueleto de código de los *Componentes de Software* que requiere ser completada. Esto puede ser realizado entonces de una de las siguientes maneras:

Siguiendo la técnica de desarrollo guiado por pruebas, en cuyo caso se procede con el contexto 2.1.8, o realizando la codificación guiada por el modelo, donde se seguiría el contexto 2.1.9. La tabla 23 muestra la definición de este contexto.

Tabla 22. Definición del contexto Codificar Componente por la estrategia guiada por modelo

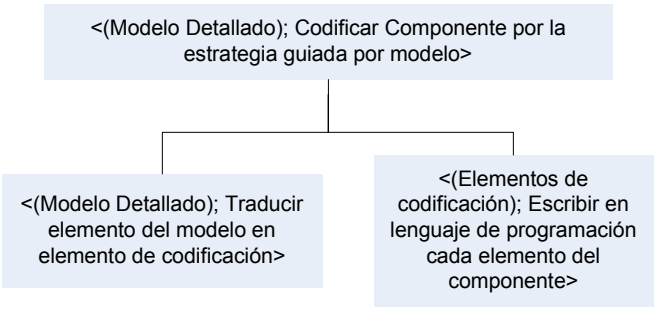
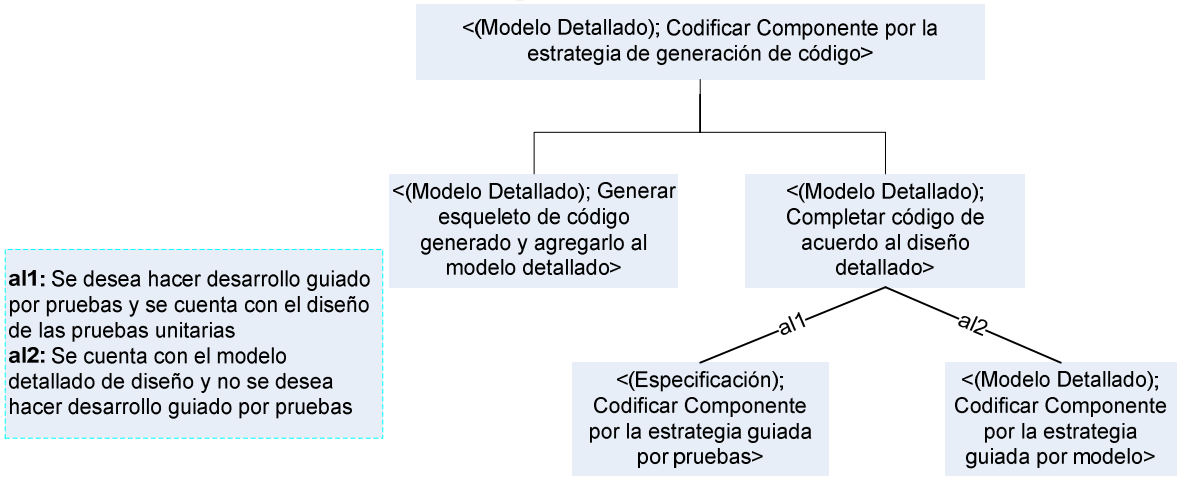
<(Modelo Detallado); Codificar Componente por la estrategia guiada por modelo>
Definición del contexto: 

Tabla 23. Definición del contexto Codificar Componente por la estrategia de generación de código

<(Modelo Detallado); Codificar Componente por la estrategia de generación de código>
Definición del contexto: 

2.1.11 Contexto: Codificar Componente por la estrategia de ajuste

Una vez que en un código ya implementado se consigue un error como resultado de una verificación o de una depuración de componente, el ajuste de dicho código es también una actividad artesanal bastante simple. Se muestra esta definición en la tabla 24.

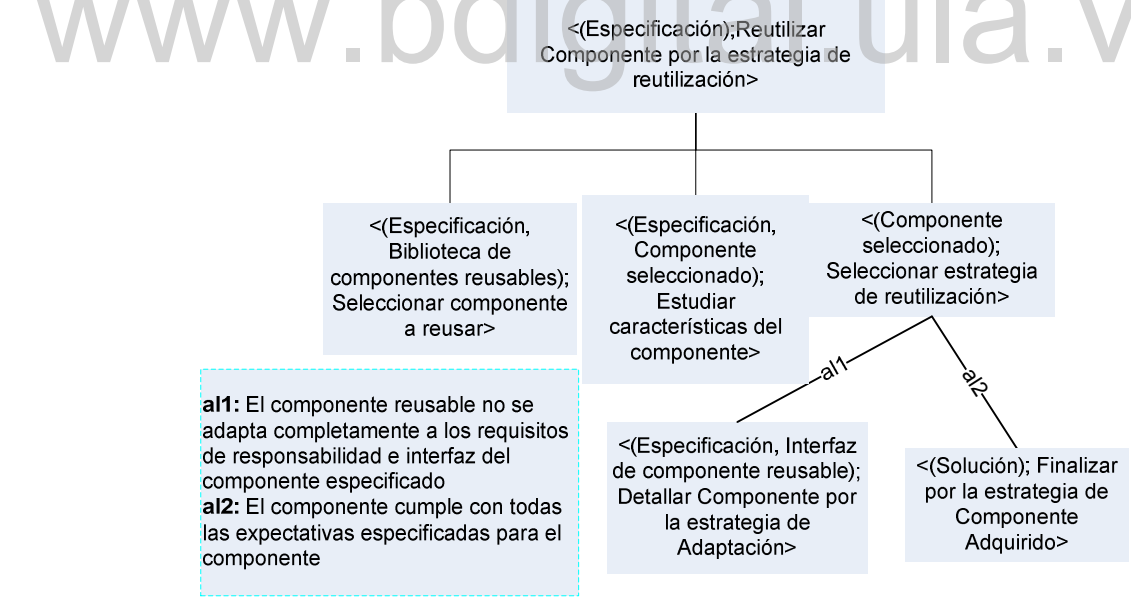
Tabla 24. Definición del contexto Codificar Componente por la estrategia de ajuste

<(Componente, con estado(Componente) = Requiere Ajuste); Codificar Componente por la estrategia de ajuste>
Definición del contexto: Una vez determinado el error, se procede al cambio del elemento de código que produce el error por una contraparte que cumpla con las especificaciones del componente, eliminando el error del mismo.

2.1.12 Contexto: Reutilizar Componente por la estrategia de reutilización

La actividad de reuso de un componente, una vez que se ha determinado la oportunidad de efectuarla, parte de la selección y el estudio de las características del componente que sería reutilizado para el aprovisionamiento de un componente o de una parte del mismo, en el contexto de la especificación de ese componente. De encontrarse un componente reusable que resulte en la satisfacción de las distintas expectativas de comportamiento e interfaz definidas para el componente, se procede a la adquisición y uso directo del componente sin modificación alguna, con lo que se puede considerar como aprovisionado este componente.

Tabla 25. Definición del contexto Reutilizar Componente por la estrategia de reutilización

<(Especificación);Reutilizar Componente por la estrategia de reutilización>
Definición del contexto: 

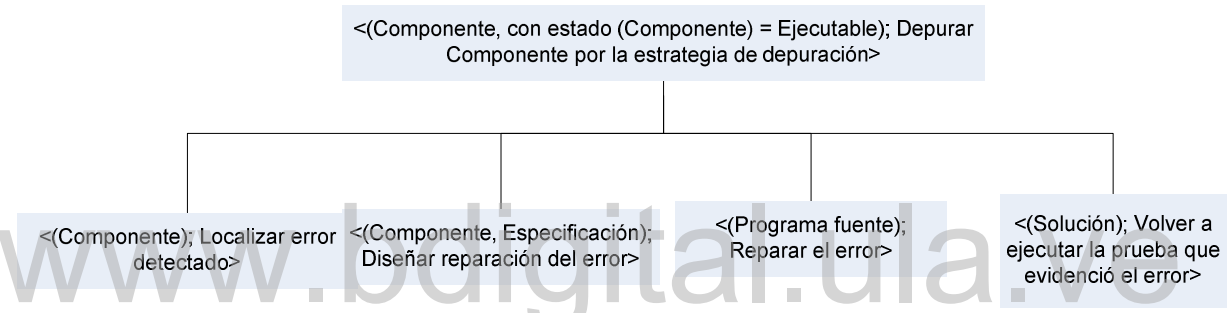
2.1.13 Contexto: Depurar Componente por la estrategia de depuración

La actividad de depuración de componente comprende la localización y corrección de los defectos que son típicamente detectados durante un proceso de pruebas. (Sommerville, 2005) describe el proceso de depuración de la siguiente manera: Se comienza por la localización del error, partiendo de alguna hipótesis acerca del comportamiento que se observa durante la ejecución del programa, o haciendo uso de herramientas de

depuración interactiva que muestran los valores intermedios de las variables del programa y una traza de las sentencias ejecutadas.

Una vez localizado el error se procede al diseño e implementación de la reparación del error, lo cual representa un cambio en la codificación del componente. Este debe nuevamente ser verificado e integrado a la solución para poder ejecutar de nuevo el caso de pruebas que evidenció el error, y así demostrar la resolución del mismo. Una buena práctica, si se sigue el desarrollo del componente guiado por pruebas, consiste en actualizar los casos de pruebas unitarias para incluir el escenario que no había sido antes contemplado y que culminó por la inclusión de un defecto en la solución. La tabla 26 muestra la definición de este contexto.

Tabla 26. Definición del contexto Depurar Componente por la estrategia de depuración

<(Componente, con estado (Componente) = Ejecutable); Depurar Componente por la estrategia de depuración>
Definición del contexto: 

2.1.14 Contexto: Verificar Componente por la estrategia de pruebas unitarias satisfechas

Tabla 27. Definición del contexto Verificar Componente por la estrategia de pruebas unitarias satisfechas

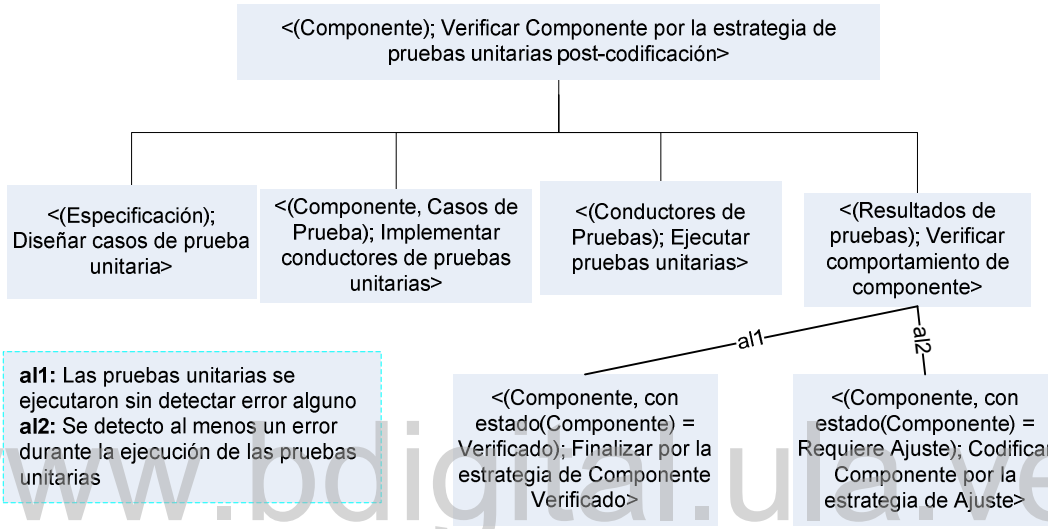
<(Componente), Verificar Componente por la estrategia de pruebas unitarias satisfechas>
Definición del contexto: Si la codificación del componente siguió la estrategia Guiada por Pruebas, el componente puede considerarse como verificado siempre que los requisitos estén bien representados en los casos de pruebas unitarias y todas las pruebas implementadas sean ejecutadas de manera satisfactoria.

2.1.15 Contexto: Verificar Componente por la estrategia de pruebas unitarias post-codificación

La verificación de componentes cuando no se ha seguido la estrategia de codificación guiada por pruebas consiste en una actividad completa, que requiere el diseño de los casos de prueba unitaria y la ejecución de los mismos para considerar el componente como verificado. Esta estrategia tiene la desventaja de desconectar las actividades de diseño detallado y codificación del componente de la necesidad de pruebas unitarias del mismo, lo cual regularmente resulta en un diseño de componente que dificulta las pruebas unitarias y hace que fácilmente se opte por descartar la verificación usando pruebas unitarias, al implicar un tiempo adicional luego de que el componente ya ha sido construido por lo que causa la sensación de ser una actividad opcional. Se comienza por el diseño de los casos de prueba unitaria, procediendo luego a la implementación y ejecución de los conductores de pruebas unitarias. Basado en los resultados de la mencionada ejecución, se determina si

debe efectuarse algún ajuste sobre el código del componente cuando se haya detectado un error en el mismo o si por el contrario se considera que el componente ha sido exitosamente verificado, y por lo tanto, se considera aprovisionado el componente.

Tabla 28. Definición del contexto Verificar Componente por la estrategia de pruebas unitarias post-codificación

<(Componente); Verificar Componente por la estrategia de pruebas unitarias post-codificación>
Definición del contexto: 

2.1.16 Contexto: Finalizar por la estrategia de componente verificado

Tabla 29. Definición del contexto Finalizar por la estrategia de componente verificado

<(Componente, con estado(Componente) = Verificado); Finalizar por la estrategia de componente verificado>
Definición del contexto: Una vez que las pruebas unitarias han sido ejecutadas satisfactoriamente, es decir, sin encontrar error en el componente, se considera que el componente ha sido verificado y se considera como aprovisionado.

2.1.17 Contexto: Finalizar por la estrategia de componente adquirido

Tabla 30. Definición del contexto Finalizar por la estrategia de componente adquirido

<(Solución); Finalizar por la estrategia de componente adquirido>
Definición del contexto: El componente se considera como aprovisionado una vez que en la reutilización de componentes se determina que el componente a reutilizar cumple con todas las especificaciones de comportamiento e interfaz indicadas para el componente solicitado.

2.1.18 Contexto: Finalizar por la estrategia de verificación diferida

Tabla 31. Definición del contexto Finalizar por la estrategia de verificación diferida

<(Solución); Finalizar por la estrategia de Componente Adquirido>
Definición del contexto: El componente se considera como aprovisionado sin realizar una verificación unitarias y previa en aquellos casos en que se determina como muy complejo o que no se disponga de la tecnología que facilitaría el desarrollo de pruebas unitarias. Esto ocurre en casos como elementos de interfaz gráfica, componentes de código que tienen alta dependencia de ambientes específicos de ejecución, o cuando se desarrolla en lenguajes de programación antiguos que no facilitan la creación de pruebas unitarias. En estos casos la verificación se realiza como parte de las pruebas de integración o en su defecto durante las pruebas funcionales.

2.2 Proveer Componente por la estrategia de atención de incidencias

Firma de la Directiva: `<(Solución); Proveer Componente por la Estrategia de Atención a Incidencias>`

Esta directiva estratégica viene a satisfacer la necesidad de los ajustes y cambios requeridos ante la detección de una no conformidad con las especificaciones (verificación) o las expectativas del cliente (validación). Las no conformidades se expresan en incidencias, las cuales son registradas en la Base de Incidencias y de allí son atendidas.

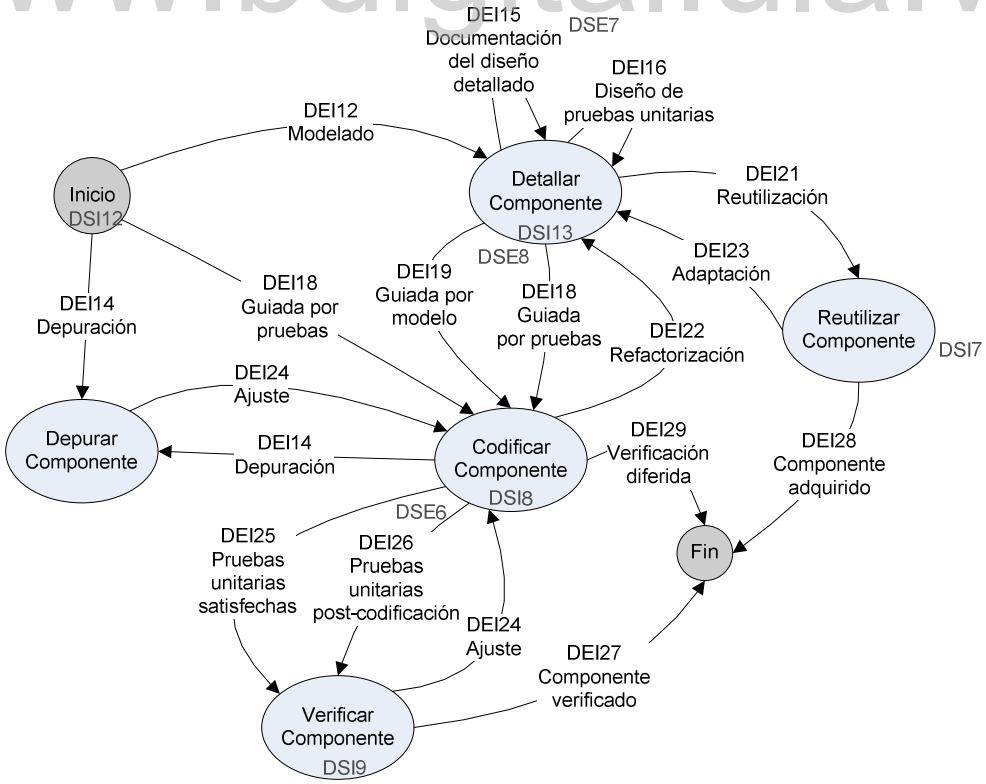


Figura 27. Mapa Local: Proveer Componente por la estrategia de Atención de Incidencias

En el mapa de la figura 27 se observa que se contemplan las mismas intenciones de la directiva estratégica 2.1, las cuales son Detallar Componente, Reutilizar Componente, Codificar Componente, Depurar Componente y Verificar Componente. No obstante, los caminos entre las intenciones aquí contenidas tienen una variación sutil pero importante. Un ejemplo de ello puede observarlo en la posibilidad de iniciar la atención de una incidencia mediante la depuración, dado que el problema ya ha sido identificado y se requiere dar con la causa que lo produce.

Las diferencias en las estrategias disponibles y los caminos que puede tomarse en esta directiva estratégica son de igual manera visibles en la tabla 32. Similar a las intenciones, las estrategias de ejecución de intención de este mapa son muy parecidas a las usadas en la directiva estratégica 2.1, razón por la que se hace referencia a las definiciones de los contextos disponibles en la sección 2.1 del presente capítulo. A continuación se muestra la tabla 32, con la definición de los caminos de la presente directiva estratégica. Así mismo, las tablas 33, 34 y 35 muestran el detalle de las directivas de selección de intención, selección de estrategia y ejecución de intención, respectivamente.

Tabla 32. Tabla de Caminos del Mapa Global de Implementación de Software

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI12	Detallar Componente	-	Modelado	DEI12
		Codificar Componente	-	Guiada por pruebas	DEI18
		Depurar Componente	-	Depuración	DEI14
Detallar Componente	DSI13	Detallar Componente	DSE7	Documentación del Diseño Detallado	DEI15
				Diseño de Pruebas Unitarias	DEI16
		Reutilizar Componente	-	Reutilización	DEI21
		Codificar Componente	DSE8	Guiada por pruebas	DEI18
				Guiada por modelo	DEI19
Depurar Componente	-	Codificar Componente	-	Ajuste	DEI24
Reutilizar Componente	DSI7	Detallar Componente	-	Adaptación	DEI23
		Fin	-	Componente adquirido	DEI28
Codificar Componente	DSI8	Detallar Componente	-	Refactorización	DEI22

		Depurar Componente	-	Depuración	DEI14
		Verificar Componente	DSE6	Pruebas unitarias satisfechas	DEI25
				Pruebas unitarias post-codificación	DEI26
		Fin	-	Verificación Diferida	DEI29
Verificar Componente	DSI9	Codificar Componente	-	Ajuste	DEI24
		Fin	-	Componente verificado	DEI27

Tabla 33. Tabla de directivas de selección de intención del mapa de Proveer Componente por la estrategia de atención de incidencias

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI7	<(Solución);Avanzar desde Reutilizar Componente>	DEI23: Seleccionar (<(Diseño Detallado);Detallar Componente por la estrategia de adaptación>) **U**	A21
		DEI28: Seleccionar (<(Solución); Finalizar por la estrategia de componente adquirido>)	A22
DSI8	<(Componente);Avanzar desde Codificar Componente>	DEI22: Seleccionar(<(Programa Fuente);Detallar Componente por la estrategia de refactorización>) **U**	A23
		DEI14: Seleccionar(<(Componente, con estado (Componente) = Ejecutable); Depurar Componente por la estrategia de depuración>) **U**	A24
		DSE6: Seleccionar(<(Solución); Avanzar hacia Verificar Componente>) **U**	A26
		DEI29: Seleccionar(<(Solución); Finalizar por la estrategia de verificación diferida>)	A25
DSI9	<(Componente); Avanzar desde Verificar Componente>	DEI24: Seleccionar(<(Componente, con estado(Componente) = Requiere Ajuste);Codificar Componente por la estrategia de ajuste>) **U**	A27
		DEI27: Seleccionar(<(Componente, con estado(Componente) = Verificado); Finalizar por la estrategia de componente verificado>)	A28
DSI12	<(Especificación); Avanzar desde Inicio>	DEI12: Seleccionar(<(Especificación); Detallar componente por la estrategia de modelado>) **U**	A29
		DEI18: Seleccionar(<(Especificación); Codificar componente por la estrategia guiada por pruebas>) **U**	A16

		DEI14: Seleccionar(<(Componente en estado(Componente) = Ejecutable); Depurar componente por la estrategia de depuración>)	A39
DSI13	<(Especificación); Avanzar desde Detallar Componente>	Seleccionar(<(Diseño Detallado); Avanzar hacia Detallar Componente>) **U**	A18
		Seleccionar(<(Diseño Detallado);Reutilizar Componente por la estrategia de Reutilización>) **U**	A19
		Seleccionar(<(Especificación); Avanzar hacia Codificar Componente>)	A20

Argumentos de Directivas

- **A16:** El entendimiento que el desarrollador tiene del componente a desarrollar es muy completo, la especificación se puede codificar directamente y ya se tienen bien conceptualizadas las pruebas unitarias que deben ser diseñadas. De igual manera, el desarrollador entiende los conceptos asociados al desarrollo orientado a pruebas.

- **A18:** Aún se detectan interrogantes de implementación en el modelo detallado que deben ser respondidas antes de comenzar la codificación, por lo que se requiere un mayor nivel de granularidad en el detalle del mencionado modelo.

- **A19:** Se ha detectado que la funcionalidad requerida para el componente siendo detallado puede ser cubierta de manera parcial o total por un componente reutilizable disponible en alguna de las bibliotecas de componentes reutilizables disponibles para el proyecto.

- **A20:** El diseño detallado está suficientemente elaborado, las especificaciones de pruebas unitarias requeridas han sido generadas, se entienden las actividades de codificación que deben ser realizadas y se han cubierto las expectativas de documentación que se requiere durante el diseño detallado del componente.

- **A21:** El componente reutilizable requiere una adaptación antes de que pueda ser utilizado en la solución en desarrollo.

- **A22:** El componente reutilizable puede ser reutilizado sin cambio alguno, y el mismo ya está disponible en los repositorios de componentes del proyecto.

- **A23:** Durante la codificación se detectó que el diseño puede ser implementado de manera más simplificada y más legible, de una manera que pueda evitar duplicación de código o que con el ajuste del diseño de un componente previo se reduce la cantidad de trabajo de implementación del componente actual.

- **A24:** Se ha detectado un comportamiento no esperado durante las pruebas iniciales del componente y no es evidente la razón del mismo.

- **A25:** Todos los elementos de trabajo asociados a la codificación han sido satisfechos y el componente ha aprobado las pruebas preliminares durante la codificación.

- **A26:** Se ha completado la codificación sin pruebas unitarias de un componente que en su especificación o en los estándares se considera como un componente muy difícil de verificar de manera unitaria.

- **A27:** Durante la verificación se ha detectado una anomalía en el componente que debe ser corregida.

- **A28:** El componente ha pasado satisfactoriamente las pruebas unitarias y cumple con los requisitos de interfaz que dicta la especificación.

- **A29:** La especificación se encuentra disponible en un formato que dificulta la generación automática del esqueleto del código o no se desea hacer uso de este tipo de facilidad.

- **A39:** La incidencia creada para el ajuste del componente explica la forma de reproducir el error causado y el componente se encuentra listo y ejecutable para realizar una depuración.

Tabla 34. Tabla de directivas de selección de estrategia del mapa de Proveer Componente por la estrategia de atención de incidencias

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE6	<(Solución); Avanzar hacia Verificar Componente>	DEI25: Seleccionar(<(Componente), Verificar Componente por la estrategia de pruebas unitarias satisfechas>) **U**	A37
		DEI26: Seleccionar(<(Componente); Verificar Componente por la estrategia de pruebas unitarias post-codificación>)	A38
DSE7	<(Modelo Detallado); Avanzar hacia Detallar Componente>	DEI15: Seleccionar(<(Diseño Detallado); Detallar Componente por la estrategia de documentación del diseño detallado>) **U**	A31
		DEI16: Seleccionar(<(Diseño Detallado); Detallar Componente por la estrategia de diseño de pruebas unitarias>) **U**	A32
DSE8	<(Especificación); Avanzar hacia Codificar Componente>	DEI18: Seleccionar(<(Especificación); Codificar Componente por la estrategia guiada por pruebas>) **U**	A34
		DEI19: Seleccionar(<(Diseño Detallado); Codificar Componente por la estrategia guiada por modelo>)	A40

Argumentos de Directivas

- **A31:** El documento de diseño detallado es un requisito explícito del proyecto, ya sea por solicitud del cliente o por que haya sido acordado de esa manera, por estándares organizacionales. Además, la complejidad de la implementación merece una explicación más detallada que la disponible en la especificación arquitectónica. Finalmente, se requiere que el diseño esté suficientemente elaborado como para que no se invierta tiempo innecesario documentando algo que tiene alta tendencia a ser cambiado, por no haber sido comprobado aún.

- **A32:** El diseño detallado ya contempla la separación de las responsabilidades entre clases, se ha completado el diseño de las interfaces del componente y se desea hacer uso del desarrollo guiado por pruebas.

- **A34:** Se desea seguir la práctica de desarrollo guiado por pruebas, se posee el diseño preliminar de las pruebas unitarias a implementar, se cuenta con un *Framework* para el desarrollo de pruebas unitarias automatizadas del componente en desarrollo y el desarrollador tiene contexto básico del desarrollo de pruebas unitarias.

- **A37:** Se ha seguido la práctica de desarrollo guiado por pruebas y las pruebas unitarias se ejecutan de manera correcta en su totalidad.

- **A38:** No se dispone de pruebas unitarias para el componente actual, sin embargo ya las actividades de codificación se concluyeron.

- **A40:** No se dispone de alguno de los requisitos del desarrollo guiado por pruebas.

Tabla 35. Tabla de directivas de ejecución de intención del mapa de Proveer Componente por la estrategia de atención de incidencias

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI12	<(Especificación); Detallar Componente por la estrategia de modelado>	Ver Contexto 2.1.1
DEI14	<(Componente, con estado (Componente) = Ejecutable); Depurar Componente por la estrategia de depuración>	Ver Contexto 2.1.13
DEI15	<(Diseño Detallado); Detallar Componente por la estrategia de documentación del diseño detallado>	Ver Contexto 2.1.3
DEI16	<(Diseño Detallado); Detallar Componente por la estrategia de diseño de pruebas unitarias>	Ver Contexto 2.1.4
DEI18	<(Especificación); Codificar Componente por la estrategia guiada por pruebas>	Ver Contexto 2.1.8
DEI19	<(Diseño Detallado); Codificar Componente por la estrategia guiada por modelo>	Ver Contexto 2.1.9
DEI21	<(Diseño Detallado); Reutilizar Componente por la estrategia de reutilización>	Ver Contexto 2.1.12
DEI22	<(Programa Fuente); Detallar Componente por la estrategia de refactorización>	Ver Contexto 2.1.6
DEI23	<(Diseño Detallado);Detallar Componente por la estrategia de adaptación>	Ver Contexto 2.1.7
DEI24	<(Componente, con estado(Componente) = Requiere Ajuste); Codificar Componente por la estrategia de ajuste>	Ver Contexto 2.1.11
DEI25	<(Componente), Verificar Componente por la estrategia de pruebas unitarias satisfechas>	Ver Contexto 2.1.14

DEI26	<(Componente); Verificar Componente por la estrategia de pruebas unitarias post-codificación>	Ver Contexto 2.1.15
DEI27	<(Componente, con estado(Componente) = Verificado); Finalizar por la estrategia de componente verificado>	Ver Contexto 2.1.16
DEI28	<(Solución); Finalizar por la estrategia de componente adquirido>	Ver Contexto 2.1.17
DEI29	<(Solución); Finalizar por la estrategia de verificación diferida>	Ver Contexto 2.1.18

2.3 Implantar Ambiente por la estrategia de preparación

Firma de la Directiva: <(Especificación); Implantar Ambiente por la estrategia de preparación>

La presente directiva estratégica está asociada a la preparación de los *Ambientes de Implementación*, en el amplio sentido utilizado a lo largo de este trabajo de grado. Una parte de ese trabajo consiste en la instalación de las herramientas que serán usadas para la implementación de los componentes (*Plataforma de Implementación*) y esta actividad está asociada a la intención Configurar Plataforma. La otra parte del trabajo de preparación del ambiente consiste en el establecimiento de las *Convenciones de Desarrollo*, que definen un marco común de estrategias, estándares y reglas comunes para el equipo de desarrollo durante la implementación de los componentes.

La figura 28 muestra el mapa local asociado a la directiva estratégica descrita, mostrando las intenciones mencionadas anteriormente.

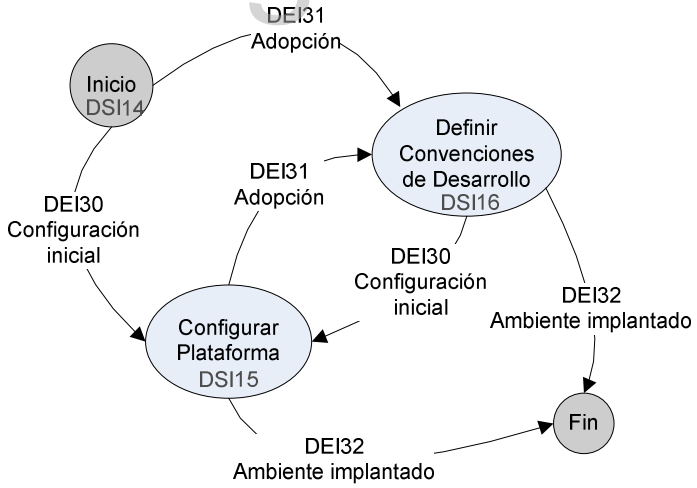


Figura 28. Mapa Local: Implantar ambiente por la estrategia de preparación

En la tabla 36 se muestran los caminos disponibles en esta directiva estratégica, indicando las referencias a las directivas que están involucradas. Luego, se pueden ver las tablas con las directivas específicas mencionadas en la tabla de caminos; la tabla 37 mostrando las directivas de selección de intención y la tabla 38 que contiene las directivas de ejecución de intención disponibles en esta directiva estratégica.

Tabla 36. Tabla de caminos del mapa local de Implantar Ambiente por la estrategia de preparación

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI14	Configurar Plataforma	-	Configuración inicial	DEI30
		Definir Convenciones de Desarrollo	-	Adopción	DEI31
Configurar Plataforma	DSI15	Definir Convenciones de Desarrollo	-	Adopción	DEI31
		Fin	-	Ambiente Implementado	DE32
Definir Convenciones de Desarrollo	DSI16	Configurar Plataforma	-	Configuración inicial	DEI30
		Fin	-	Ambiente Implementado	DE32

Tabla 37. Tabla de directivas de selección de intención del mapa Implantar Ambiente por la estrategia de preparación

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI14	<(Solución); Avanzar desde Inicio>	DEI30: Seleccionar(<(Solución); Configurar Plataforma por la estrategia de configuración inicial>)	A41
		DEI31: Seleccionar(<(Solución); Definir Convenciones de Desarrollo por la estrategia de adopción>) **U**	A42
DSI15	<(Solución); Avanzar desde Configurar Plataforma>	DEI31: Seleccionar(<(Solución); Definir Convenciones de Desarrollo por la estrategia de adopción>) **U**	A42
		DEI32: Seleccionar(<(Ambiente de Implementación); Finalizar por la estrategia de ambiente implantado>)	A43
DSI16	<(Solución); Avanzar desde Definir Convenciones de Desarrollo>	DEI30: Seleccionar(<(Solución); Configurar Plataforma por la estrategia de configuración inicial>) **U**	A41
		DEI32: Seleccionar(<(Ambiente de Implementación); Finalizar por la estrategia de ambiente implantado>)	A43

Argumentos de Directivas

- **A41:** La plataforma de desarrollo mínima no está completamente configurada, y falta la configuración de las herramientas básicas necesarias para dar soporte al Proceso de Implementación

- **A42:** Las convenciones de desarrollo no se encuentran completamente definidas, falta definición de estándares de construcción o de diseño de componentes

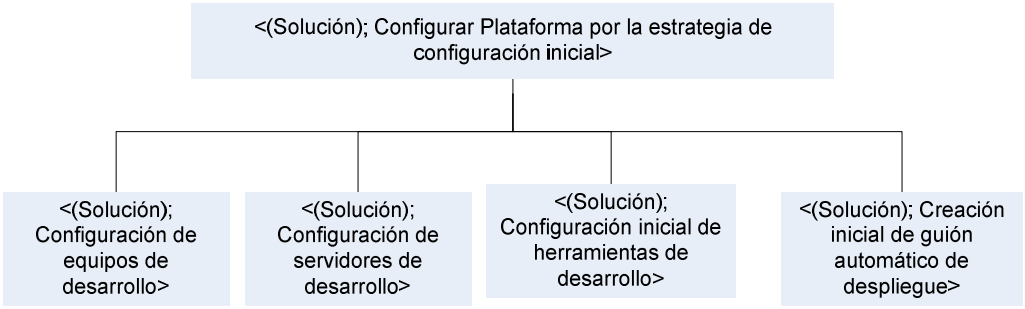
- **A43:** Tanto la plataforma de desarrollo como las convenciones de desarrollo se encuentran en un nivel suficiente de definición como para comenzar a realizar las actividades del Proceso de Implementación

Tabla 38. Tabla de directivas de ejecución de intención del mapa Implantar Ambiente por la estrategia de preparación

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI30	`<(Solución); Configurar Plataforma por la estrategia de configuración inicial>`	Ver Contexto 2.3.1
DEI31	`<(Solución); Definir Convenciones de Desarrollo por la estrategia de adopción>`	Ver Contexto 2.3.2
DEI32	`<(Ambiente de Implementación); Finalizar por la estrategia de ambiente implantado>`	Ver Contexto 2.3.3

2.3.1 Contexto: Configurar Plataforma por la estrategia de configuración inicial

Tabla 39. Definición del contexto Configurar Plataforma por la estrategia de configuración inicial

<(Solución); Configurar Plataforma por la estrategia de configuración inicial>
Definición del contexto: 

Como se explica en el modelo de productos, la Plataforma de Desarrollo contempla los sistemas hardware y software que soportan el Proceso de Implementación. La configuración inicial consiste en asegurar que se cuenta con las herramientas que agilizarán la implementación de los componentes, entre los que se cuentan los equipos de desarrollo, servidores de desarrollo y las herramientas mencionadas en el modelo de productos. Especial mención merece el guión de despliegue automático de la aplicación, que será de gran utilidad para prácticas como la integración continua. La tabla 39 muestra la definición de este contexto.

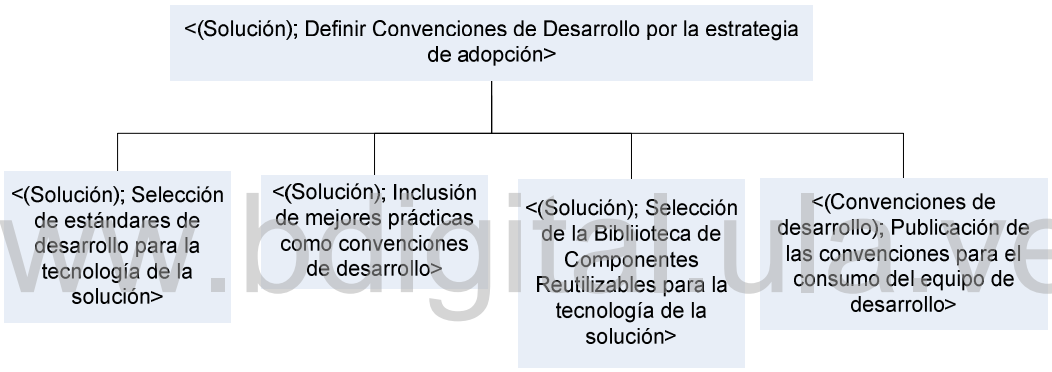
2.3.2 Contexto: Definir Convenciones de Desarrollo por la estrategia de adopción

Las convenciones de desarrollo, como se describió en el modelo de productos, involucran aquellos acuerdos explícitos que establecen un marco común para las decisiones de implementación tomadas día a día por los desarrolladores. Aquí se contemplan los Estándares de Construcción, que buscan la promoción de la

codificación por parte del todo el equipo de la misma manera, y la definición de la o las Bibliotecas de Componentes Reutilizables.

La estrategia de Adopción busca tomar para el equipo los estándares que puedan aplicar para la tecnología seleccionada, partiendo de definiciones y convenciones públicas generalmente aceptadas y la inclusión en las convenciones de las mejores prácticas reconocidas por la comunidad de Tecnologías de Información (TI). Esta actividad incluiría los siguientes pasos: Selección de estándares de desarrollo basados en la tecnología, adopción de buenas prácticas como convenciones de desarrollo, selección de la Biblioteca de Componentes Reutilizables para la tecnología de la solución y publicación de estas convenciones para el consumo del todo el equipo de desarrollo. La tabla 40 muestra la especificación gráfica de este contexto.

Tabla 40. Definición del contexto Definir Convenciones de Desarrollo por la estrategia de adopción

<(Solución); Definir Convenciones de Desarrollo por la estrategia de adopción>
Definición del contexto: 

2.3.3 Contexto: Finalizar por la estrategia de ambiente implantado

Tabla 41. Definición del contexto Finalizar por la estrategia de ambiente implantado

<(Ambiente de Implementación); Finalizar por la estrategia de ambiente implantado>
Definición del contexto: Una vez que tanto la Plataforma como las Convenciones de Desarrollo se encuentran establecidas, se considera que el ambiente ha sido exitosamente implantado y se procede a la implementación de los componentes de la solución.

2.4 Implantar Ambiente por la estrategia de ajuste de ambiente

Firma de la Directiva: `<(Especificación); Implantar Ambiente por la estrategia de ajuste de ambiente>`

Esta directiva estratégica viene a satisfacer la necesidad que pudiese existir de modificar el *Ambiente de Implementación*, ya sea en su *Plataforma de Implementación* o en las *Convenciones de Desarrollo*. Tal necesidad puede surgir como parte de la adaptación del ambiente al uso de un componente reutilizable o por

alguna característica del ambiente que se determinó durante el diseño detallado de algún *Componente de Software*. La figura 29 muestra la representación de mapa de esta directiva.

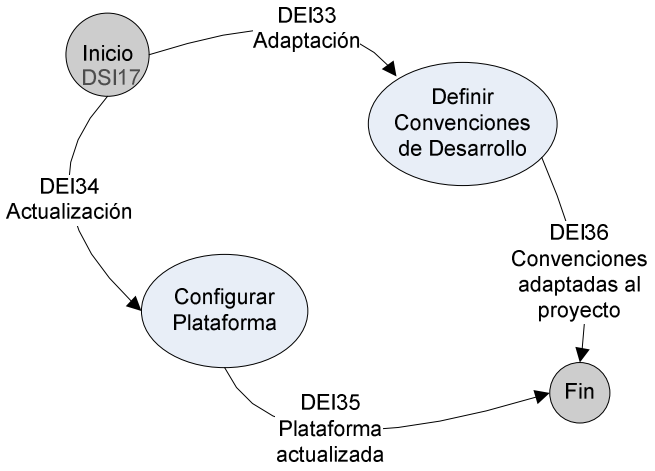


Figura 29. Mapa Local: Implantar ambiente por la estrategia de ajuste

Se observa en el mapa que esta directiva esta formada por las mismas intenciones asociadas a la directiva estratégica 2.3: *Configurar Plataforma* y *Definir Convenciones de Desarrollo*. El cambio principal ocurre en las estrategias asociadas a las intenciones. En el caso de *Configurar Plataforma*, la única estrategia posible es la de actualización, representando esto algún posible cambio de versión de alguno de los elementos que conforman la *Plataforma de Desarrollo*, o incluso la instalación de un nuevo elemento software de la plataforma. Por su parte, la estrategia asociada a *Definir Convenciones de Desarrollo* es la de adaptación, que primordialmente se refiere a los ajustes en las convenciones adoptadas y acordadas por el equipo de desarrollo a alguna particularidad del entorno del proyecto en manos. La tabla 42 muestra los caminos posibles del mapa de esta directiva estratégica. Las tablas 43, 44 listan las directivas de selección de intención y ejecución de intención que forman parte del mapa de la figura 29.

Tabla 42. Tabla de caminos del mapa local de Implantar Ambiente por la estrategia de ajuste de ambiente

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI17	Definir Convenciones de Desarrollo	-	Adaptación	DEI33
		Configurar Plataforma	-	Actualización	DEI34
Definir Convenciones de Desarrollo	-	Fin	-	Convenciones adaptadas al proyecto	DEI36
Configurar Plataforma	-	Fin	-	Plataforma actualizada	DEI35

Tabla 43. Tabla de directivas de selección de intención del mapa Implantar ambiente por la estrategia de ajuste

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI17	<(Solución); Avanzar desde Inicio>	DEI34: Seleccionar(<(Solución); Configurar Plataforma por la estrategia de actualización>) **U**	A44
		DEI33: Seleccionar(<(Solución); Definir Convenciones de Desarrollo por la estrategia de adaptación>)	A45

Argumentos de Directivas

- **A44:** Se requiere un cambio en la configuración de la Plataforma de Desarrollo, sea asociado al Ambiente mínimo de desarrollo o a las herramientas de soporte del Proceso de Implementación
- **A45:** Se ha detectado un elemento que debe ser considerado una convención de desarrollo, por lo que se adaptan las convenciones originales para incluir el nuevo elemento.

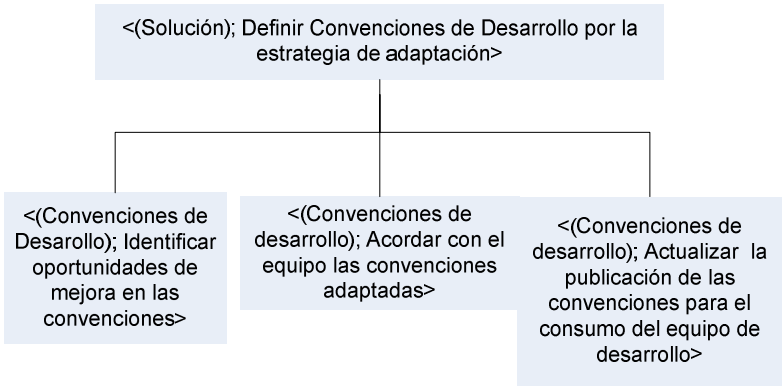
Tabla 44. Tabla de directivas de selección de intención del mapa Implantar ambiente por la estrategia de ajuste

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI33	<(Solución); Definir Convenciones de Desarrollo por la estrategia de adaptación>	Ver Contexto 2.4.1
DEI34	<(Solución); Configurar Plataforma por la estrategia de actualización>	Ver Contexto 2.4.2
DEI35	<(Ambiente de Implementación); Finalizar por la estrategia de plataforma actualizada>	Ver Contexto 2.4.3
DEI36	<(Ambiente de Implementación); Finalizar por la estrategia de convenciones adaptadas al proyecto>	Ver Contexto 2.4.4

2.4.1 Contexto: Definir Convenciones de Desarrollo por la estrategia de adaptación

En cualquier momento del desarrollo algún miembro del equipo puede realizar una propuesta de mejora a las convenciones usadas por el equipo basado en algún tipo de compatibilidad ya sea con la tecnología, la solución o con la cultura del equipo, lo que pudiese estar representando un obstáculo para la efectividad o eficiencia en la implementación. Siempre que se verifique la compatibilidad de la propuesta de mejora con las condiciones de la solución o el proyecto de desarrollo, y que se logre un acuerdo entre los miembros del equipo de desarrollo, se puede realizar la adaptación de las convenciones de desarrollo, lo cual se debe oficializar con la actualización de las convenciones de desarrollo que fueron inicialmente publicadas. La tabla 45 contiene la especificación de este contexto.

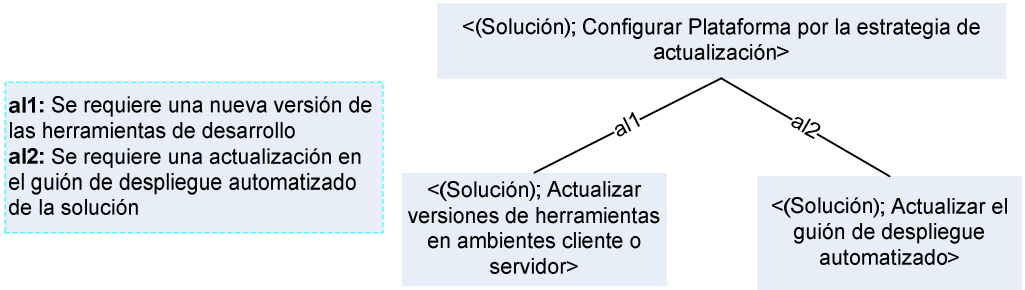
Tabla 45. Definición del contexto Definir Convenciones de Desarrollo por la estrategia de adaptación

<(Solución); Definir Convenciones de Desarrollo por la estrategia de adaptación>
Definición del contexto: 

2.4.2 Contexto: Configurar Plataforma por la estrategia de actualización

Una actualización de la plataforma puede darse tanto en la forma de instalación de nuevas versiones de las herramientas clientes o servidor como en forma de actualización del guión de despliegue automatizado de la solución. Estas situaciones se muestran en la definición del contexto en la tabla 46.

Tabla 46. Definición del contexto Configurar Plataforma por la estrategia de actualización

<(Solución); Configurar Plataforma por la estrategia de Actualización>
Definición del contexto: 

2.4.3 Contexto: Finalizar por la estrategia de plataforma actualizada

Tabla 47. Definición del Finalizar por la estrategia de plataforma actualizada

<(Ambiente de Implementación); Finalizar por la estrategia de plataforma actualizada>
Definición del contexto: El proceso se considera finalizado cuando la actualización de la plataforma ha sido exitosamente completada.

2.4.4 Contexto: Finalizar por la estrategia de convenciones adaptadas al proyecto

Tabla 48. Definición del contexto Finalizar por la estrategia de Convenciones Adaptadas al Proyecto

<(Ambiente de Implementación); Finalizar por la estrategia de convenciones adaptadas al proyecto>
Definición del contexto: Al haber un acuerdo en las mejoras requeridas a las convenciones de desarrollo y la atención de las incidencias que puedan ocurrir asociadas dichas convenciones, se considera que el proceso culmina de manera exitosa.

2.5 Integrar Componente por la estrategia de Integración por Lotes

Firma de la Directiva: `<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la Estrategia de Integración en Lotes>`

La directiva estratégica actual se asocia a la intención de orquestar y enlazar un *Componente de Software* recién aprovisionado tanto a la *Plataforma de Desarrollo* como a los otros *Componentes de Software* que hayan sido provistos anteriormente, para que el nuevo componente cubra las responsabilidades que se le asignaron por diseño. La estrategia aquí descrita se asocia a la integración de componentes realizada sin el uso de técnicas o herramientas que permitan la automatización de este proceso. Por ello, utiliza estrategias y técnicas primordialmente manuales.

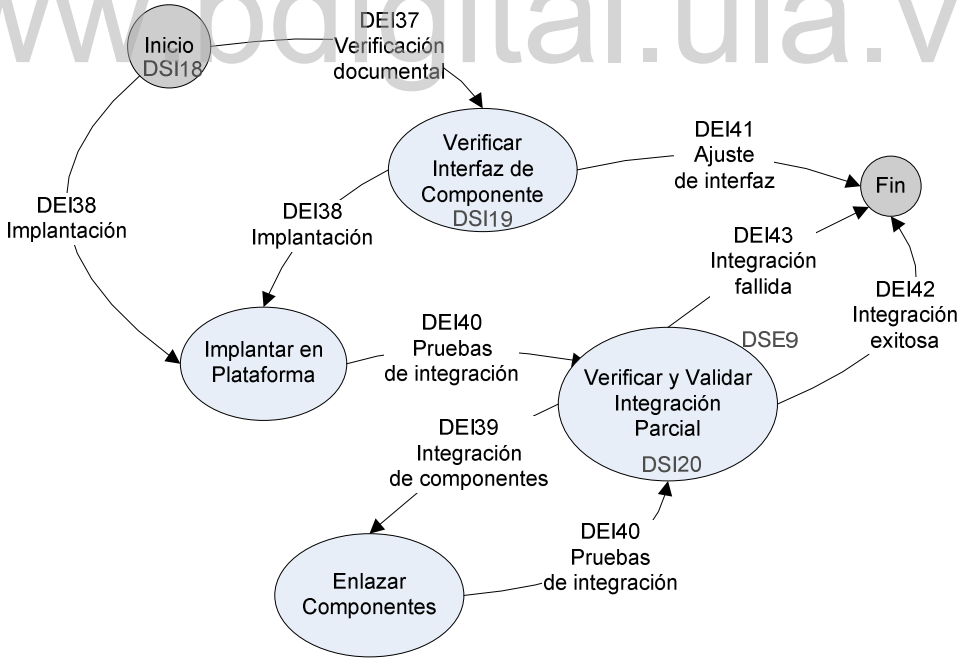


Figura 30. Mapa Local: Integrar Componente por la estrategia de integración por lotes

La figura 30 muestra el mapa asociado a esta directiva estratégica. Se observan aquí las intenciones de 1) Verificar Interfaz del Componente, la cual busca determinar por vía documental o por revisión manual detallada de la *Interfaz del Componente* la perfecta compatibilidad de los puntos de conexión del componente a ser integrado con sus contrapartes, sean otros *Componentes de Software* o parte de la *Plataforma de Desarrollo*; 2)

Implantar en Plataforma, que consiste en el despliegue primario del componente en su plataforma de ejecución, tal como indique la especificación y el diseño del componente, con lo que se busca evaluar y verificar la compatibilidad del componente con la mencionada plataforma; 3) Enlazar Componentes, que realiza de manera puntual la conexión del *Componente de Software* con sólo uno de sus contrapartes componentes. El que se realice este enlace con un solo componente a ser conectado permite que la última de las intenciones del mapa 4) Verificar y Validar Integración Parcial, logre evaluar el estado del sistema completo una vez que se ajusta una variable del sistema, en este caso, la conexión de un componente nuevo con cada uno de sus interrelacionados.

En la tabla 49 se pueden observar los caminos mostrados en el mapa de la Figura 30. Las tablas 50, 51 y 52, muestran a su vez las directivas de selección de intención, selección de estrategia y ejecución de intención disponibles en la presente directiva estratégica.

Tabla 49. Tabla de caminos del mapa local de Integrar Componente por la estrategia de integración por lotes

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI18	Verificar Interfaz del Componente	-	Verificación documental	DEI37
		Implantar en Plataforma	-	Implantación	DEI38
Verificar Interfaz del Componente	DSI19	Implantar en Plataforma	-	Implantación	DEI38
		Fin	-	Ajuste de interfaz	DEI41
Implantar en Plataforma	-	Verificar y Validar Integración Parcial	-	Pruebas de integración	DEI40
Verificar y Validar Integración Parcial	DSI20	Enlazar Componentes	-	Integración de componentes	DEI39
		Fin	DSE9	Integración exitosa	DEI42
				Integración fallida	DEI43
Enlazar Componentes	-	Verificar y Validar Integración Parcial	-	Pruebas de integración	DEI40

Tabla 50. Tabla de directivas de selección de intención del mapa Integrar Componente por la estrategia de integración por lotes

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI18	`<(Solución); Avanzar desde Inicio>`	DEI37: Seleccionar(`<(Especificación); Verificar Interfaz de Componente por estrategia de verificación documental>`)	A46

		DEI38: Seleccionar(<(Componente); Implantar en Plataforma por estrategia de implantación>) **U**	A47
DSI19	<(Solución); Avanzar desde Verificar Interfaz de Componente>	DEI38: Seleccionar(<(Componente); Implantar en Plataforma por estrategia de implantación>) **U**	A48
		DEI41: Seleccionar(<(Solución); Finalizar por estrategia de ajuste de interfaz>)	A49
DSI20	<(Solución); Avanzar desde Verificar Integración Parcial>	DEI39: Seleccionar(<(Producto); Enlazar Componentes por estrategia de integración de componentes>) **U**	A50
		DSE9: Seleccionar(<(Solución); Avanzar hacia Fin>)	A51

Argumentos de Directivas

- **A46:** La interfaz del componente es de suma importancia para la integración con sistemas externos o para proveer servicios a terceros.
- **A47:** La interfaz de uso del componente no es crítica para aplicaciones o servicios externos.
- **A48:** La verificación de la interfaz fue exitosa y la integración puede continuar sin problemas
- **A49:** Se detectó un problema en la interfaz del componente y se requiere su ajuste antes de que pueda integrarse exitosamente dicho componente.
- **A50:** Se ha guardado el resultado de las integraciones parciales que se han realizado hasta el momento y aun faltan componentes por ser integrados con el componente actual.
- **A51:** Se han terminado los componentes por integrar o se han detectado errores que no permiten completar las pruebas en las integraciones parciales

Tabla 51. Tabla de directivas de selección de estrategia del mapa Integrar Componente por la estrategia de integración por lotes

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE9	<(Solución); Avanzar hacia Fin>	DEI42: Seleccionar(<(Solución); Finalizar por la estrategia de integración exitosa>) **U**	A52
		DEI43: Seleccionar(<(Solución); Finalizar por la estrategia de integración fallida>)	A53

Argumentos de Directivas

- **A52:** La integración con los componentes previamente aprovisionados se realizó en su totalidad y sin arrojar error alguno
- **A53:** Una o más integraciones parciales arrojaron errores que deben ser corregidos para reintentar la integración de componentes luego.

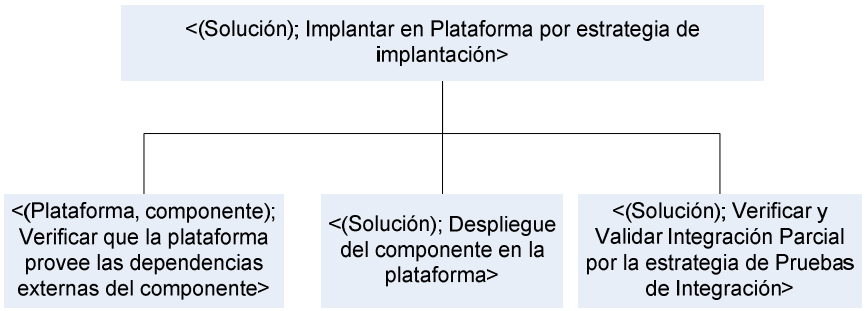
Tabla 52. Tabla de directivas de ejecución de intención del mapa Integrar Componente por la estrategia de integración por lotes

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI37	<(Especificación); Verificar Interfaz de Componente por estrategia de verificación documental>	Ver Contexto 2.5.2
DEI38	<(Componente); Implantar en Plataforma por estrategia de implantación>	Ver Contexto 2.5.1
DEI39	<(Producto); Enlazar Componentes por estrategia de integración de componentes>	Ver Contexto 2.5.4
DEI40	<(Solución); Verificar y Validar Integración Parcial por la estrategia de pruebas de integración>	Ver Contexto 2.5.3
DEI41	<(Solución); Finalizar por estrategia de ajuste de interfaz>	Ver Contexto 2.5.5
DEI42	<(Solución); Finalizar por la estrategia de integración exitosa>	Ver Contexto 2.5.6
DEI43	<(Solución); Finalizar por la estrategia de integración fallida>	Ver Contexto 2.5.7

2.5.1 Contexto: Implantar en Plataforma por estrategia de implantación

La implantación en plataforma consiste en la colocación del componente aprovisionado en un ambiente con condiciones similares al especificado para el funcionamiento real de la solución en desarrollo. Para un componente sin dependencias especiales del ambiente, consiste en un entorno donde como mínimo se dispone de las librerías, *frameworks* y servicios necesarios para la ejecución de componentes implementados usando la tecnología seleccionada, como sistema operativo, procesador, máquinas virtuales, servidores de aplicación, librerías dinámicas, entre otros. En el caso que el componente haga uso de servicios especiales provisto por un componente externo a la aplicación, dicho componente debe estar disponible como parte de la plataforma.

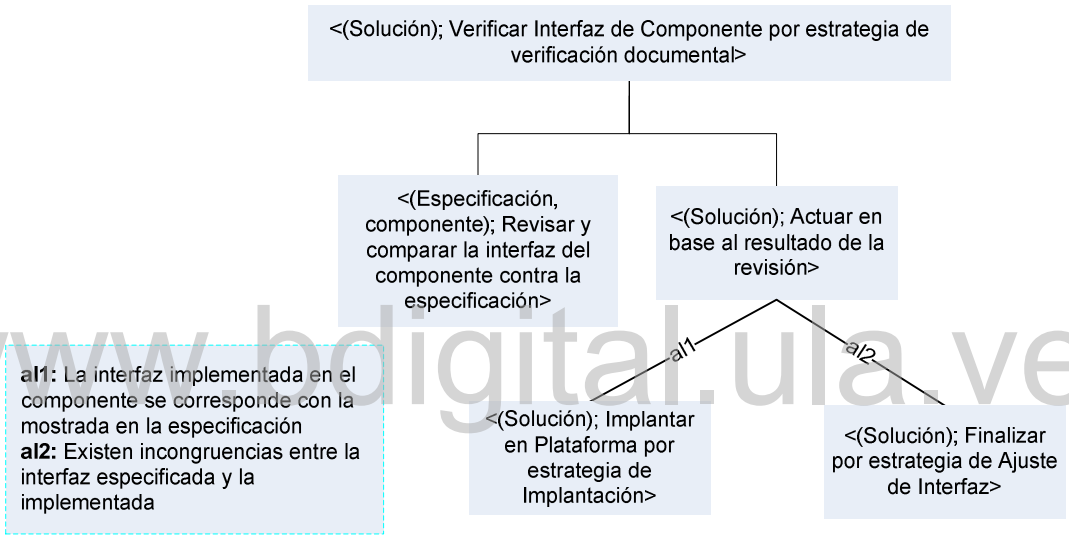
Tabla 53. Definición del contexto Implantar en Plataforma por estrategia de implantación

<(Solución); Implantar en Plataforma por estrategia de implantación>
Definición del contexto: 

El primer paso de esta actividad consiste en la verificación de la disponibilidad de las dependencias externas del componente en la plataforma. Luego se procede al despliegue del componente o de un compilado de la solución que contenga el componente a ser desplegado, para luego proceder a la verificación y validación del funcionamiento del componente con su ambiente de ejecución. La tabla 53 muestra la definición de este contexto.

2.5.2 Contexto: Verificar Interfaz de Componente por estrategia de verificación documental

Tabla 54. Definición del contexto Verificar Interfaz de Componente por estrategia de verificación documental

<(Solución); Verificar Interfaz de Componente por estrategia de verificación documental>
Definición del contexto: 

La actividad de verificación de interfaz consiste en la auditoría de la interfaz implementada en comparación con lo documentado en la especificación del componente. Se hace una sencilla revisión visual del componente y basado en el cumplimiento de la especificación de la interfaz, se procede a implantar en la plataforma si se verifica el cumplimiento o se detiene el proceso de integración al encontrar una incongruencia entre lo implementado y lo especificado a nivel de interfaz.

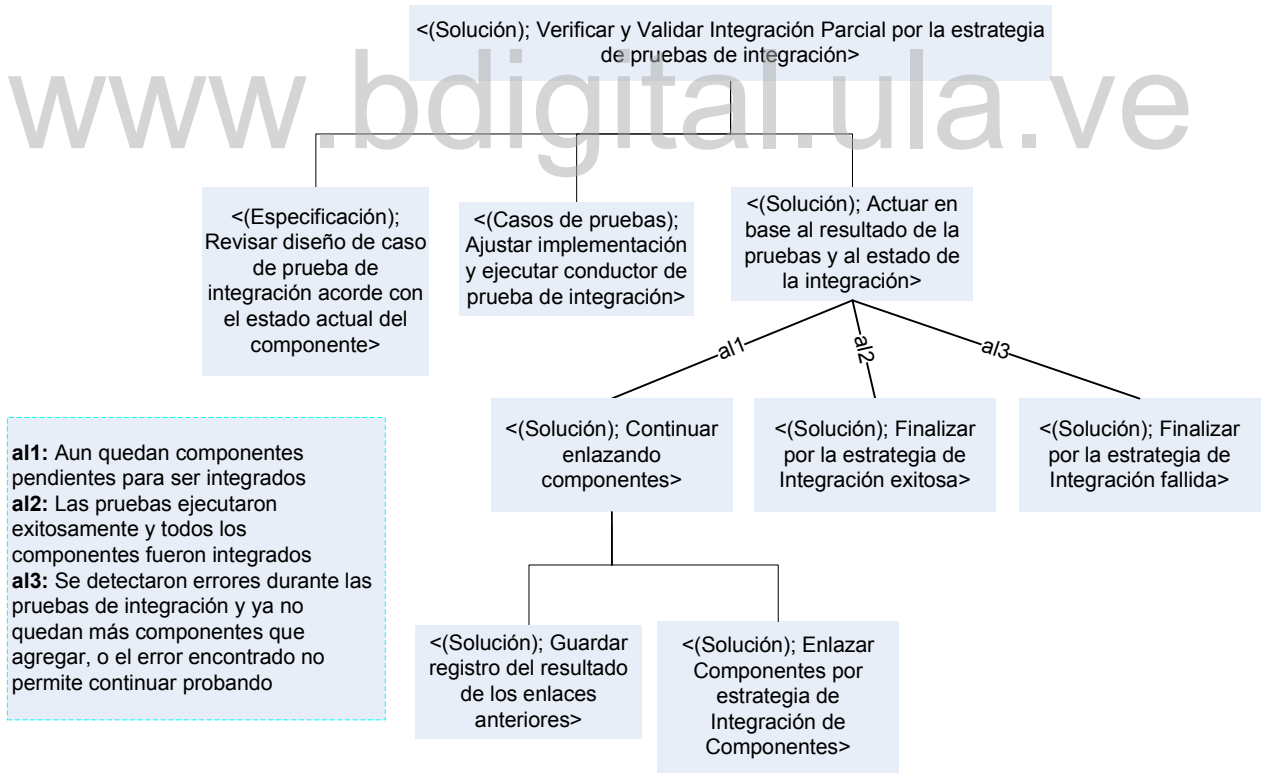
2.5.3 Contexto: Verificar y Validar Integración Parcial por la estrategia de pruebas de integración

Con cada implantación y componente agregado a la integración, se realiza la verificación y validación de la integración a través de pruebas de integración. El objetivo de este tipo de pruebas consiste en agregar componentes de manera incremental con el fin de detectar rápidamente que vértice en una integración multi-nodos puede estar fallando, cosa que sería difícil de determinar si se procede a integrar todo y luego hacer pruebas integrales.

Durante cada ciclo de integración podría requerirse la creación o el ajuste de la implementación de los conductores de pruebas de integración, pero generalmente va a reutilizarse las pruebas que han sido usadas en los ciclos de integración anteriores. Sin embargo, debe verificarse que el diseño de los casos de pruebas de

los ciclos anteriores aun se corresponda con la especificación de los componentes y cualquier ajuste que sea requerido en los conductores se efectúe. Una vez realizado este ajuste, se procede a la ejecución de las pruebas, y dependiendo del resultado y del estado actual de la integración se pueden tomar 3 decisiones distintas: Si aun faltan componentes por integrar, se registra el resultado de la integración parcial (sea exitosa o no) y se procede a la integración del próximo componente; si todas las pruebas fueron satisfactorias y ya todos los componentes fueron integrados, entonces finaliza el proceso indicando que la integración fue exitosa; y si se integraron todos los componentes y se detectó al menos un error durante las pruebas de integración parciales, se reportan las incidencias de errores en integración asociado al respectivo nodo integrado en ese momento y se finaliza el proceso de integración indicando que la integración fue fallida. Como principio general se intenta continuar en las pruebas, de manera de detectar tantos errores como sea posible durante un ciclo de integración. No obstante, si alguno de los errores no permite continuar avanzando en las pruebas de integración, se termina indicando las pruebas que no pudieron ser realizadas debido al error. La especificación de este contexto puede ser observada en la tabla 55.

Tabla 55. Definición del Verificar Integración Parcial por la estrategia de pruebas de integración

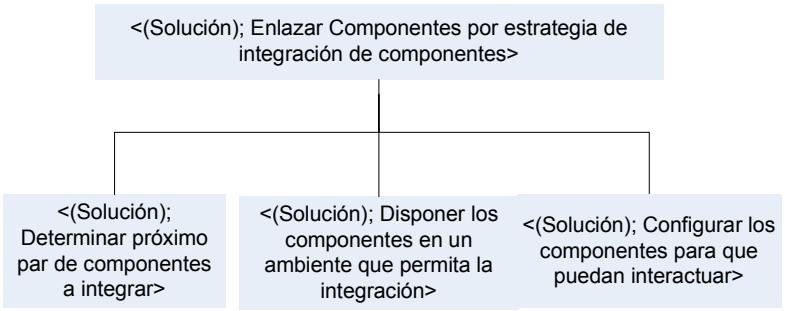
<(Solución); Verificar y Validar Integración Parcial por la estrategia de pruebas de integración>
Definición del contexto: 

2.5.4 Contexto: Enlazar Componentes por estrategia de integración de componentes

El proceso de enlace de componentes consiste en agregar a la integración un componente a la vez, por lo que se mantiene un orden y un estado actual de la integración. Una vez que se determine el próximo par de

componentes a integrarse, se ubican los componentes en ambientes que puedan ser accedidos por el otro componente, y se procede a la configuración de cada componente para proceder a la integración. La tabla 56 muestra la definición formal de este contexto.

Tabla 56. Definición del contexto Enlazar Componentes por estrategia de integración de componentes

<(Solución); Enlazar Componentes por estrategia de integración de componentes>
Definición del contexto: 

2.5.5 Contexto: Finalizar por estrategia de ajuste de interfaz

Tabla 57. Definición del contexto Finalizar por estrategia de ajuste de interfaz

<(Solución); Finalizar por estrategia de ajuste de interfaz>
Definición del contexto: Al detectarse una incongruencia en la verificación de interfaz, se procede a dar por terminado el proceso de integración, y se realiza el contexto <Integrar Componente, Proveer Componente, Estrategia de Atención de Incidencias>.

2.5.6 Contexto: Finalizar por la estrategia de integración exitosa

Tabla 58. Definición del Finalizar por la estrategia de integración exitosa

<(Solución); Finalizar por la estrategia de integración exitosa>
Definición del contexto: Luego de culminarse la integración de todos los componentes de manera exitosa, se considera como terminada la integración de componentes.

2.5.7 Contexto: Finalizar por la estrategia de integración fallida

Tabla 59. Definición del contexto Finalizar por la estrategia de integración fallida

<(Solución); Finalizar por la estrategia de integración fallida>
Definición del contexto: Al ejecutarse la mayor cantidad posible de pruebas y se detectan diversos errores, o cuando un solo error en las pruebas evita la posibilidad de seguir probando, se procede a finalizar el proceso de integración indicando

el fallo en la integración y ejecutando el contexto `<Integrar Componente, Proveer Componente, Estrategia de Atención de Incidencias>`.

2.6 Integrar Componente por la estrategia de integración continua

Firma de la Directiva: `<(Componente con estado (Componente) = Aprovisionado); Integrar Componente por la estrategia de integración continua>`

El proceso de Integración Continua consiste en la integración puntual de cada mínimo cambio que se realice en los componentes, como parte de la culminación del aprovisionamiento de un componente. Se requieren algunos elementos necesarios para la ejecución de este proceso como una práctica común en un equipo de desarrollo. Entre los prerrequisitos se encuentran el contar con una suite de pruebas automatizadas para las pruebas de integración, así como un guión automatizado de despliegue de los componentes para rápidamente realizar el proceso de integración y de verificación y validación de la integración resultante. Esta práctica puede no sólo realizarse con cada cambio que haya sido actualizado en el sistema de control de versiones usado en el desarrollo, sino que puede hacerse integraciones nocturnas del código realizado el día anterior, ejecutando una suite de pruebas más completas. La figura 31 muestra el mapa local de la integración de componentes por la estrategia de Integración Continua.

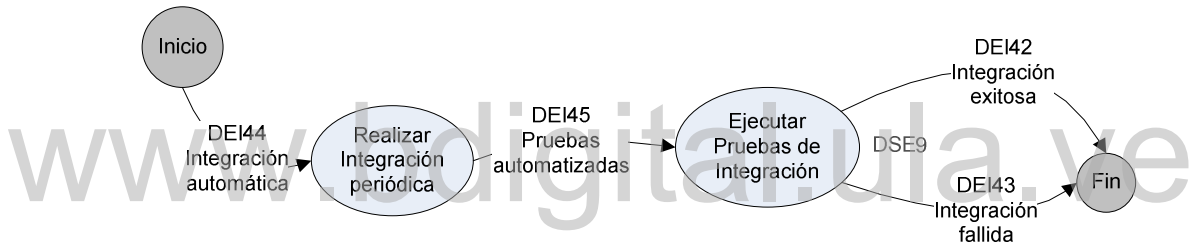


Figura 31. Mapa Local: Integrar Componente por la estrategia de integración continua

La intención de Realizar Integración Periódica se logra a través de la integración automática que consiste en el despliegue automatizado con un *Guión Automatizado de Despliegue* de la solución. Posterior al este despliegue automatizado, se tiene la intención de Ejecutar Pruebas de Integración, donde se procede a la ejecución del conjunto de *Conductores de Pruebas Automatizadas de Integración*, las cuales notificarán también de manera automática acerca de la ejecución exitosa o fallida del proceso de Integración. La tabla 60 muestra la tabla de caminos de esta directiva estratégica, y en las tablas 60 y 61 las directivas asociadas.

Tabla 60. Tabla de caminos del mapa local de Integrar Componente por la estrategia de integración continua

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	-	Realizar Integración Periódica	-	Integración automática	DEI44
Realizar Integración Periódica	-	Ejecutar Pruebas de Integración	-	Pruebas automatizadas	DEI45

Ejecutar Pruebas de Integración	-	Fin	DSE9	Integración exitosa	DEI42
				Integración fallida	DEI43

Tabla 61. Tabla de directivas de selección de estrategia del mapa Integrar Componente por la estrategia de integración continua

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE9	`<(Solución); Avanzar hacia Fin>`	DEI42: Seleccionar(`<(Solución); Finalizar por la estrategia de Integración exitosa>`) **U**	A52
		DEI43: Seleccionar(`<(Solución); Finalizar por la estrategia de Integración fallida>`)	A53

Argumentos de Directivas

- **A52:** La integración con los componentes previamente aprovisionados se realizó en su totalidad y sin arrojar error alguno
- **A53:** Una o más integraciones parciales arrojaron errores que deben ser corregidos para reintentar la integración de componentes luego.

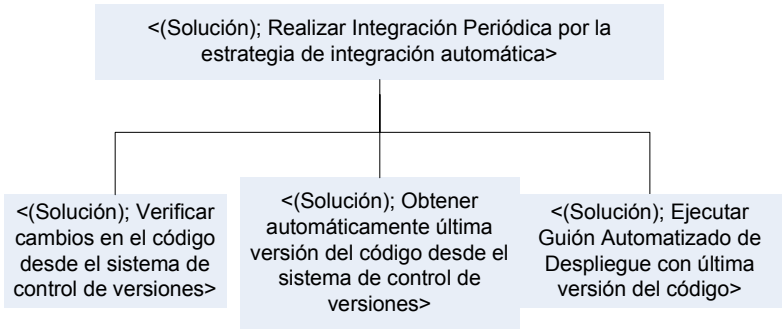
Tabla 62. Tabla de directivas de ejecución de intención del mapa Integrar Componente por la estrategia de integración continua

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI44	`<(Solución); Realizar Integración Periódica por la estrategia de integración automática>`	Ver Contexto 2.6.1
DEI45	`<(Solución); Ejecutar Pruebas de Integración por la estrategia de pruebas automatizadas>`	Ver Contexto 2.6.2
DEI42	`<(Solución); Finalizar por la estrategia de integración exitosa>`	Ver Contexto 2.5.7
DEI43	`<(Solución); Finalizar por la estrategia de integración fallida>`	Ver Contexto 2.5.8

2.6.1 Contexto: Realizar Integración Periódica por la estrategia de integración automática

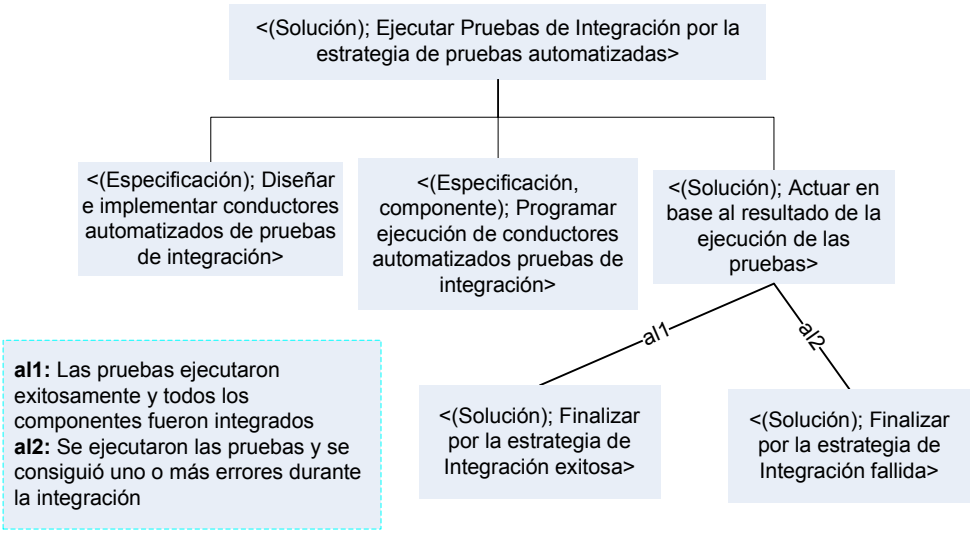
La Integración Periódica se ejecuta de manera programada usando el *Servidor de Integración Continua*, que se encarga de verificar cualquier cambio que exista en el *Sistema de Control de Versiones* usado en el desarrollo, de existir un cambio procede a obtener la última versión del código y procede a ejecutar el *Guión Automatizado de Despliegue*, lo cual se encargará de la creación de un compilado de los distintos componentes de la solución y los desplegará en el ambiente usado para la integración.

Tabla 63. Definición del contexto Realizar Integración Periódica por la estrategia de integración automática

<(Solución); Realizar Integración Periódica por la estrategia de integración automática>
Definición del contexto: 

2.6.2 Contexto: Ejecutar Pruebas de Integración por la estrategia de pruebas automatizadas

Tabla 64. Definición del contexto Ejecutar Pruebas de Integración por la estrategia de pruebas automatizadas

<(Solución); Ejecutar Pruebas de Integración por la estrategia de pruebas automatizadas>
Definición del contexto: 

Las pruebas de integración en este escenario deben ser diseñadas e implementadas a priori y se ubican de manera accesible para que el servidor de Integración Continua pueda usarlas y ejecutarlas durante la verificación de la integración. Haciendo uso de la misma programación que realiza la integración de manera automatizada, se procede a programar la ejecución de las pruebas de integración y en base al resultado arrojado por la ejecución de las pruebas se procede a finalizar el proceso de manera exitosa, si las pruebas no

detectaron error alguno, o de manera fallida, si por el contrario se encuentran uno o más errores durante la integración de los componentes. La tabla 64 muestra la definición formal de este contexto.

2.7 Actualizar Documento Operativo

Firma de la Directiva: `<(Especificación); Actualizar documento operativo por la estrategia de Documentación>`

La actualización de los documentos operativos por parte del equipo de desarrollo es el motivo principal de la actual directiva estratégica. Si bien se ha recomendado la disminución de los documentos generados como productos de trabajo como parte del Proceso de Implementación de Software, estos documentos operativos son considerados parte de la *Solución* completa y por lo tanto son productos entregables al cliente final. Como se persigue la consistencia en la agilidad, eficiencia y la creación de productos de calidad, la creación y actualización de los documentos operativos de la *Solución* deben involucrar la participación del equipo de desarrollo para lograr tales metas.

Como se menciona en el modelo de productos descrito en el Capítulo IV, los documentos operativos o manuales a contemplar en este Proceso de Implementación son el *Manual de Uso*, el *Manual de Mantenimiento* y el *Manual de Instalación*. Para la respectiva actualización o creación de cada uno de estos documentos se encuentra disponible una intención en el mapa local descrito en la figura 32 y asociado a la presente directiva estratégica. La actualización o creación de cada uno de estos manuales está también basado en el establecimiento de tales manuales como parte del producto final, es decir, si el proyecto indica que uno o todos los mencionados documentos operativos son esperados por el cliente como entregables del proyecto.

La tabla 65 muestra los caminos posibles de esta directiva estratégica. Por su parte, las tablas 66 y 67 muestran los respectivos listados de las directivas de selección y ejecución de intención del mapa mostrado en la figura 32.

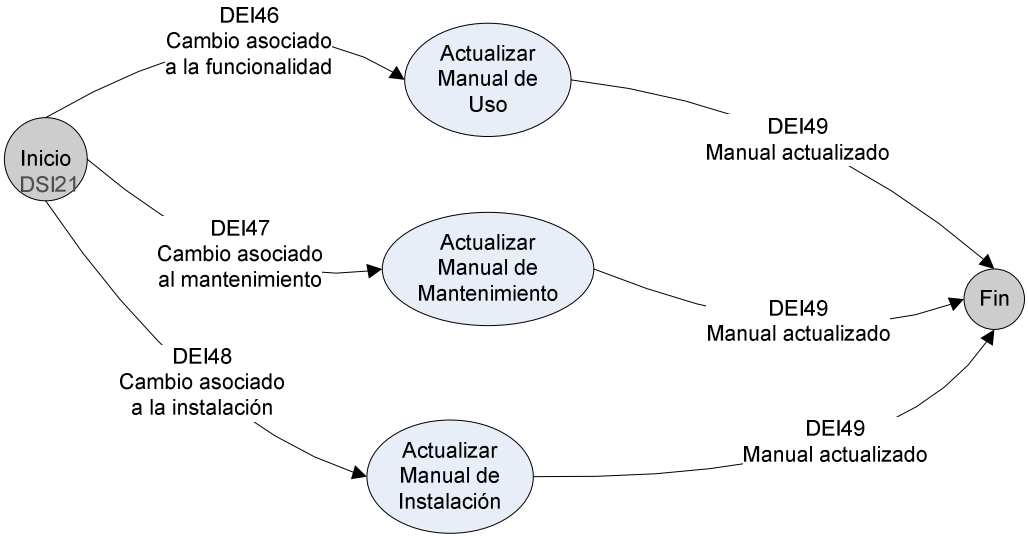


Figura 32. Mapa Local: Actualizar Documento Operativo

Tabla 65. Tabla de caminos del mapa local de Actualizar Documento Operativo

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	DSI21	Actualizar Manual de Uso	-	Cambio asociado a la funcionalidad	DEI46
		Actualizar Manual de Mantenimiento	-	Cambio asociado al mantenimiento	DEI47
		Actualizar Manual de Instalación	-	Cambio asociado a la instalación	DEI48
Actualizar Manual de Uso	-	Fin	-	Manual actualizado	DEI49
Actualizar Manual de Mantenimiento	-	Fin	-	Manual actualizado	DEI49
Actualizar Manual de Instalación	-	Fin	-	Manual actualizado	DEI49

Tabla 66. Tabla de directivas de selección de intención del mapa Actualizar Documento Operativo

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI21	`<(Solución); Avanzar desde Inicio>`	DEI46: Seleccionar(`<(Solución); Actualizar Manual de Uso por la estrategia de cambio asociado a la funcionalidad>`) **U**	A54
		DEI47: Seleccionar(`<(Solución); Actualizar Manual de Mantenimiento por la estrategia de cambio asociado al mantenimiento>`) **U**	A55
		DEI48: Seleccionar(`<(Solución); Actualizar Manual de Instalación por la estrategia de cambio asociado a la instalación>`)	A56

Argumentos de Directivas

- **A54:** El plan de implementación actual indica la creación o actualización del *Manual de Uso*, y existen cambios que deban ser reflejados en el mismo manual
- **A55:** El plan de implementación actual indica la creación o actualización del *Manual de Mantenimiento*, y existen cambios que deban ser reflejados en el mismo manual
- **A56:** El plan de implementación actual indica la creación o actualización del *Manual de Instalación*, y existen cambios que deban ser reflejados en el mismo manual

Tabla 67. Tabla de directivas de selección de intención del mapa Actualizar Documento Operativo

Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI46	<(Solución); Actualizar Manual de Uso por la estrategia de cambio asociado a la funcionalidad>	Ver Contexto 2.7.1
DEI47	<(Solución); Actualizar Manual de Mantenimiento por la estrategia de cambio asociado al mantenimiento>	Ver Contexto 2.7.2
DEI48	<(Solución); Actualizar Manual de Instalación por la estrategia de cambio asociado a la instalación>	Ver Contexto 2.7.3
DEI49	<(Solución); Finalizar por la estrategia de manual actualizado>	Ver Contexto 2.7.4

2.7.1 Contexto: Actualizar Manual de Uso por la estrategia de cambio asociado a la funcionalidad

Tabla 68. Definición del contexto Actualizar Manual de Uso por la estrategia de cambio asociado a la funcionalidad

<(Solución); Actualizar Manual de Uso por la estrategia de cambio asociado a la funcionalidad>
Definición del contexto: Al existir un cambio asociado a la manera en que el uso de la solución varía, se procede a la actualización del *Manual de Uso*.

2.7.2 Contexto: Actualizar Manual de Mantenimiento por la estrategia de cambio asociado al mantenimiento

Tabla 69. Definición del contexto Actualizar Manual de Mantenimiento por la estrategia de cambio asociado al mantenimiento

<(Solución); Actualizar Manual de Mantenimiento por la estrategia de cambio asociado al mantenimiento>
Definición del contexto: El *Manual de Mantenimiento* registra elementos asociados a la estructura, configuración interna de los componentes de software, proceso de generación de despliegues entre otros. Cualquier cambio en este sentido debe ser expresado en el manual de mantenimiento del sistema.

2.7.3 Contexto: Actualizar Manual de Instalación por la estrategia de cambio asociado a la instalación

Tabla 70. Definición del contexto Actualizar Manual de Instalación por la estrategia de cambio asociado a la instalación

<(Solución); Actualizar Manual de Instalación por la estrategia de cambio asociado a la instalación>
Definición del contexto: Al existir un cambio asociado a la manera en que la instalación de la solución varía, se procede a

la actualización del *Manual de Instalación*.

2.7.4 Contexto: Finalizar por la estrategia de manual actualizado

Tabla 71. Definición del contexto Finalizar por la estrategia de manual actualizado

<(Solución); Finalizar por la estrategia de Manual actualizado>
Definición del contexto: Una vez que la actualización requerida se haya realizado sobre el manual correspondiente, se considera que el proceso ha terminado satisfactoriamente.

2.8 Asegurar Calidad del Código

Firma de la Directiva: `<(Plan de Verificación y Validación); Asegurar calidad del código por la estrategia de Verificación Técnica>`

La presente y última directiva estratégica descrita como parte del Proceso de Implementación, está asociada a las actividades de Verificación y Validación que persiguen evaluar la calidad del código fuente de los componentes aprovisionados por este proceso y realizar las notificaciones correspondientes a las no conformidades detectadas en el código fuente. Se tienen como eventos disparadores de subproceso descrito en esta directiva estratégica la ocurrencia regular en el Plan de Implementación de alguna de las actividades de inspección y aseguramiento de la calidad del código fuente. Entre estas actividades, se recomienda la aplicación de Inspecciones técnicas regulares y muy especialmente de la Revisión de Pares, por parte de todos los miembros del equipo de desarrollo.

Las intenciones que conforman esta directiva estratégica son 1) Inspeccionar código, que busca la revisión individual o colectiva de una selección del código fuente de *Componentes de Software* de la *Solución*; y 2) Notificar Incidencias, que tiene como objetivo la comunicación efectiva de las no conformidades en forma de incidencias a las que se le puede hacer planificación y seguimiento para su corrección como parte de las actividades de implementación.

La figura 33 muestra la representación de mapa de la directiva estratégica descrita.

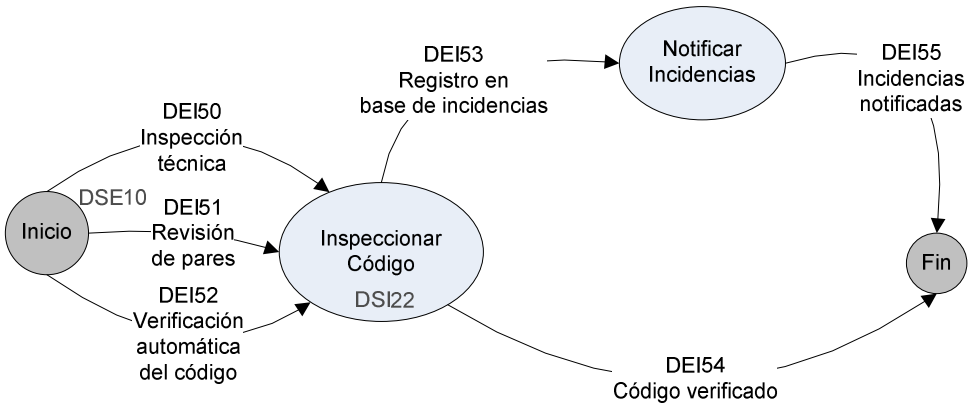


Figura 33. Mapa Local: Asegurar Calidad del Código

La tabla 72 muestra los caminos que pueden ser encontrados en el mapa mostrado anteriormente. Por su parte, en las tablas 73 y 74 se listan las directivas de selección de estrategias y ejecución de intención, respectivamente.

Tabla 72. Tabla de caminos del mapa local de Asegurar Calidad del Código

Intención fuente	Directiva de Selección de Intención	Intención destino	Directiva de Selección de Estrategia	Estrategia	Directiva de Ejecución de Intención
Inicio	-	Inspeccionar Código	DSE10	Inspección técnica	DEI50
				Revisión de pares	DEI51
				Verificación automática del código	DEI52
Inspeccionar Código	DSI22	Notificar Incidencias	-	Registro en base de incidencias	DEI53
		Fin	-	Código verificado	DEI54
Notificar Incidencias	-	Fin	-	Incidencias notificadas	DEI55

Tabla 73. Tabla de directivas de selección de intención del mapa Asegurar Calidad del Código

Directivas de Selección de Intención			
Dir.	**Firma**	**Opciones**	**Arg.**
DSI22	<(Solución);Avanzar desde Inspeccionar Código>	DEI53: Seleccionar(<(Solución); Notificar Incidencias por la estrategia de registro en base de incidencias>) **U**	A57
		DEI54: Seleccionar(<(Solución); Finalizar por la estrategia de código verificado>)	A58

Argumentos de Directivas

- **A57:** Durante la inspección de código se encontró una o más no conformidades con la especificación, los estándares o las buenas prácticas que deben ser notificadas
- **A58:** Ninguna no conformidad fue descubierta, por lo que el producto puede considerarse verificado en la calidad de sus programas fuentes

Tabla 74. Tabla de directivas de selección de estrategia del mapa Asegurar Calidad del Código

Directivas de Selección de Estrategia			
Dir.	**Firma**	**Opciones**	**Arg.**
DSE10	<(Solución); Avanzar hacia Inspeccionar Código>	DEI50: Seleccionar(<(Solución); Inspeccionar Código por la estrategia de inspección técnica>) **U**	A59

		DEI51: Seleccionar(<(Solución); Inspeccionar Código por la estrategia de revisión de pares>)	A60
		DEI52: Seleccionar(<(Solución); Inspeccionar Código por la estrategia de verificación automática de código>)	A61

Argumentos de Directivas

- **A59:** Cuando el equipo de desarrolladores no tenga la pericia suficiente para detectar faltas a las especificaciones, los estándares o las buenas prácticas y se cuente con un personal externo capaz de ejecutar dicha inspección técnica
- **A60:** El equipo de desarrolladores se encuentra en capacidad de detectar oportunidades de mejora en la calidad de los programas fuentes para corregir faltas a las especificaciones, los estándares o las buenas prácticas.
- **A61:** Cuando se cuenta con herramientas que realizan la verificación del código fuente para asegurar el uso de las mejores prácticas de codificación y notifican de manera automática las no conformidades detectadas durante la evaluación del código fuente. Es necesario que se contemple un esquema de ejecución automática de este tipo de herramientas, como por ejemplo considerarlo uno de los pasos a seguir durante la programación de herramientas de Integración Continua.

Tabla 75. Tabla de directivas de ejecución de intención del mapa Asegurar Calidad del Código

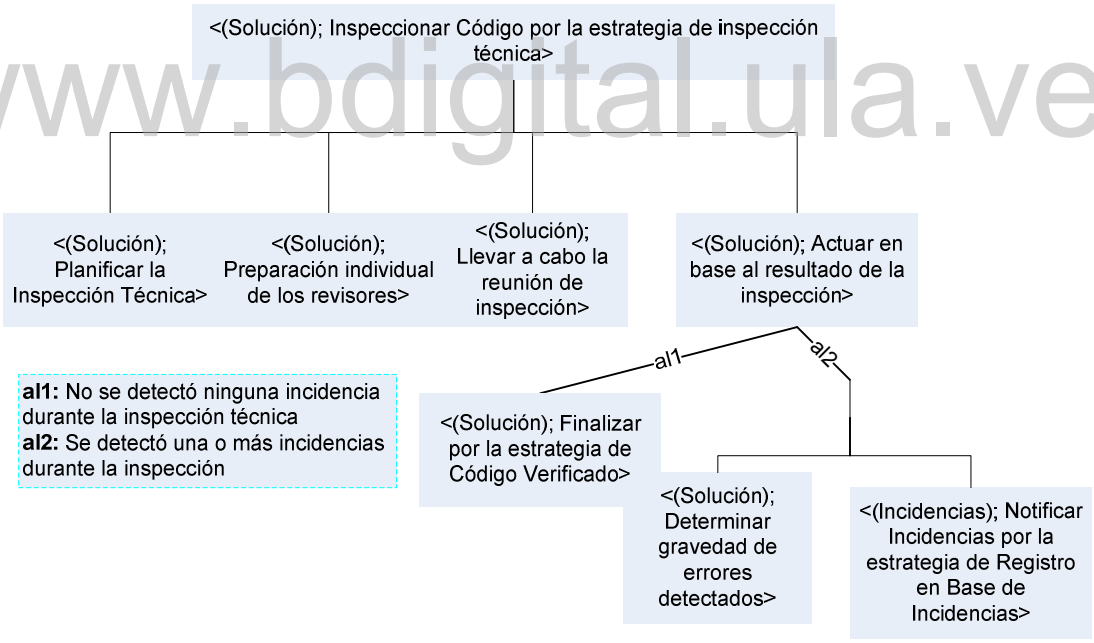
Directivas de Ejecución de Intención		
Dir.	**Firma**	**Realización**
DEI50	<(Solución); Inspeccionar Código por la estrategia de inspección técnica>	Ver Contexto 2.8.1
DEI51	<(Solución); Inspeccionar Código por la estrategia de revisión de pares>	Ver Contexto 2.8.2
DEI52	<(Solución); Inspeccionar Código por la estrategia de verificación automática de código>	Ver Contexto 2.8.3
DEI53	<(Solución); Notificar Incidencias por la estrategia de registro en base de incidencias>	Ver Contexto 2.8.4
DEI54	<(Solución); Finalizar por la estrategia de incidencias notificadas>	Ver Contexto 2.8.5
DEI55	<(Solución); Finalizar por la estrategia de código verificado>	Ver Contexto 2.8.6

2.8.1 Contexto: Inspeccionar Código por la estrategia de inspección técnica

La inspección técnica o revisión técnica es una actividad que garantiza la calidad del software llevada a cabo por los ingenieros del software cuyos objetivos son el descubrir errores en la función, la lógica o la implementación de cualquier representación del software, el verificar que el software bajo revisión alcanza sus requisitos, garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos, así

como conseguir un software desarrollado de forma uniforme (Pressman, 2002). Sommerville acota que si bien puede revisarse cualquier representación del software, generalmente se centran en el código fuente (Sommerville, 2005). La inspección técnica comienza con la planificación, la selección de los revisores y el aseguramiento de la disponibilidad y completitud del material y sus especificaciones. Una vez asignados los revisores, estos deben prepararse de manera individual para la inspección técnica haciendo una revisión por adelantado y detectando errores y oportunidades de mejora que sean expuestas durante la reunión de inspección. Posteriormente se procede a la realización de la reunión de inspección donde el desarrollador del programa fuente procede a hacer el recorrido mostrando el producto de trabajo en revisión mientras los revisores exponen sus observaciones acerca de los defectos encontrados. Al finalizar la reunión los participantes deben decidir la gravedad de los errores detectados, donde las alternativas son aceptar el producto sin modificaciones posteriores, rechazar el producto debido a los serios errores encontrados, los cuales deben ser corregidos y donde debe ejecutarse una revisión o aceptar el producto provisionalmente al encontrar errores menores que deben ser corregidos. Finalmente, se procede a la notificación de las incidencias que hayan sido encontradas durante la inspección.

Tabla 76. Definición del contexto Inspeccionar Código por la estrategia de inspección técnica

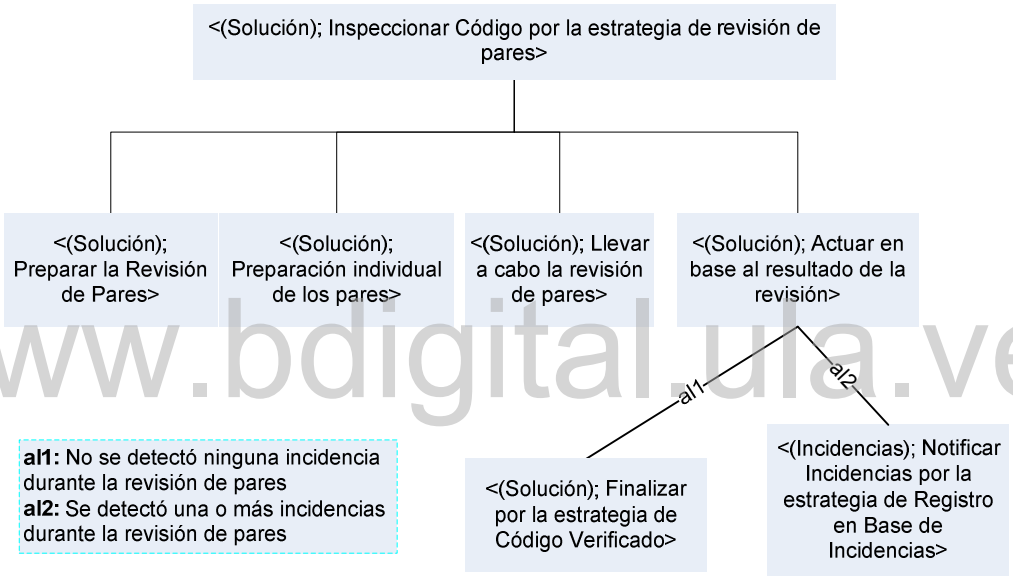
<(Solución); Inspeccionar Código por la estrategia de inspección técnica>
Definición del contexto: 

2.8.2 Contexto: Inspeccionar Código por la estrategia de revisión de pares

La Revisión de Pares involucra una revisión metódica de los productos de trabajo por parte de los pares de los productores para identificar defectos para su remoción y para recomendar otros cambios que sean necesarios (*CMMI Product Team*, 2006). Difiere de la Inspección Técnica en que se hace de manera explícita con otros desarrolladores en el caso del código fuente, quienes pueden tener contexto del trabajo a realizar y que pueden aprovecharse de esta técnica de verificación para entender y aprender sobre la implementación de los componentes realizados por otros miembros del equipo. Se comienza por la planificación y preparación de la

revisión de pares, asignando los roles de la revisión y el alcance que tendrá la misma. Los participantes de la revisión deben prepararse con anticipación para la revisión, y luego se procede a efectuar la reunión donde se exponen los programas fuentes objetos de la revisión de pares y los pares mencionan las oportunidades de mejoras encontradas así como los problemas que pudiese presentarse con la implementación actual de los componentes expuestos. Finalmente, basado en el resultado de la revisión y si se encuentran incidencias, se procede a la notificación de las mismas. En caso de no encontrarse incidencia alguna, el proceso termina satisfactoriamente. La tabla 64 presenta la definición de este contexto.

Tabla 77. Definición del contexto Verificar Inspeccionar Código por la estrategia de revisión de pares

<(Solución); Inspeccionar Código por la estrategia de revisión de pares>
Definición del contexto: 

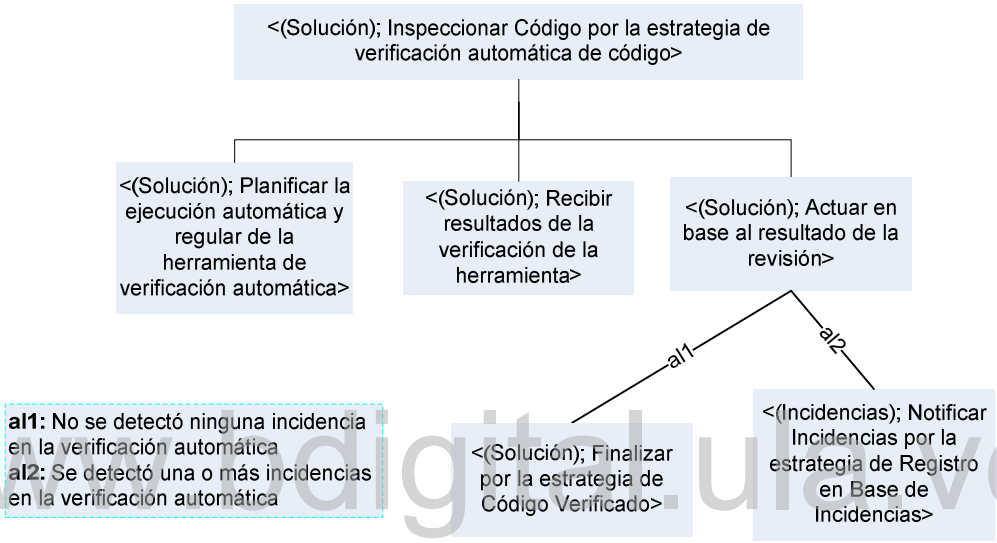
2.8.3 Contexto: Inspeccionar Código por la estrategia de verificación automática de código

La verificación automática de código requiere la existencia de herramientas de chequeo de uso de mejores prácticas de codificación, que esté disponible para el lenguaje de programación utilizado en la codificación de los *Componentes de Software*. Tales herramientas, consideradas descendientes de la herramienta *lint* usada para la verificación del código C en UNIX de Bell Labs V7, realizan lo que se conoce como análisis estático de código fuente (Ayewah et al, 2008) y son de gran utilidad para asegurar la calidad del código fuente de *Componentes de Software*. En la actualidad existen herramientas de este tipo para lenguajes de programación como *Java, C, C++, C#, Delphi, JavaScript, Ada, Perl, Objective-C*, entre otros.

Una vez que se cuenta con la herramienta de análisis estático de código para el lenguaje de programación en uso, la práctica de inspección de código usando tal herramienta parte de la planificación de la ejecución regular y automática de esta herramienta, bien sea como parte del proceso de compilación de la aplicación o como uno de los pasos ejecutados en el *Guión Automatizado de Despliegue* de la aplicación. Siguiendo esta última técnica, se tendría que la verificación automática puede ser realizada cada vez que se realiza la Integración Continua automatizada de la aplicación, y las no conformidades serían recibidas de la misma manera que se

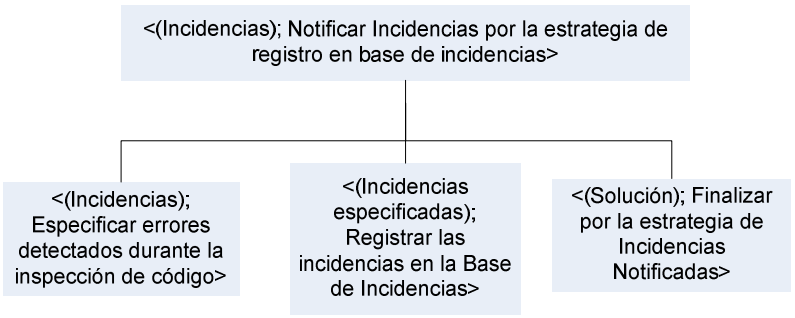
reciben los errores encontrados durante la ejecución de las pruebas de integración. Si se consigue alguna no conformidad detectada por la herramienta y considerada como tal en las *Convenciones de Desarrollo* establecidas por el equipo, se procede a la notificación de la incidencia. La tabla 78 contiene la definición formal del contexto descrito.

Tabla 78. Definición del contexto Verificar Inspeccionar Código por la estrategia de revisión de pares

<(Solución); Inspeccionar Código por la estrategia de verificación automática de código>
Definición del contexto: 

2.8.4 Contexto: Notificar Incidencias por la estrategia de registro en base de incidencias

Tabla 79. Definición del contexto Notificar Incidencias por la estrategia de registro en base de incidencias

<(Incidencias); Notificar Incidencias por la estrategia de registro en base de incidencias>
Definición del contexto: 

La notificación de incidencias una vez que se detectan en la inspección de código se hace haciendo uso de una Base de Incidencias, que bien podría ser un sistema de seguimiento de incidencias o algún mecanismo similar. El proceso consiste en la especificación de las incidencias para su atención por parte del equipo de desarrollo,

el registro de estas incidencias para su ejecución y seguimiento, y se finaliza el proceso de aseguramiento de la calidad del código una vez culminada la notificación de incidencias.

2.8.5 Contexto: Finalizar por la estrategia de incidencias notificadas

Tabla 80. Definición del contexto Finalizar por la estrategia de incidencias notificadas

<(Solución); Finalizar por la estrategia de incidencias notificadas>
Definición del contexto: Una vez que las incidencias asociadas a la calidad del código han sido notificadas, se considera que el proceso de aseguramiento de la calidad del código ha concluido.

2.8.6 Contexto: Finalizar por la estrategia de código verificado

Tabla 81. Definición del contexto Finalizar por la estrategia de código verificado

<(Solución); Finalizar por la estrategia de código verificado>
Definición del contexto: Una vez concluido el proceso de inspección del código sin detectar incidencias que atenten contra la calidad del código, se procede a terminar el proceso de aseguramiento de la calidad del código.

3 Conclusiones de este capítulo

En este capítulo se presentaron de manera detallada los fragmentos situacionales del Proceso de Implementación de Software ágil y disciplinado, siguiendo la notación descrita en el capítulo III. Se observó que las situaciones expresadas en los contextos estaban principalmente soportadas por el Modelo de Productos definido en el capítulo IV.

Puede observarse que la notación de fragmentos situacionales de procesos puede ser de gran valor dado que hace evidente las razones y los motivos para ejecutar una u otra acción. Sin embargo, su lectura es bastante densa sin la herramienta apropiada, y su manejo sólo de manera documental es realmente cuesta arriba.

CAPITULO VI: Instanciación del Modelo Situacional de Implementación de Software

Luego de la definición de los fragmentos de productos y de los fragmentos de procesos para el método situacional de implementación de software, es parte fundamental de este trabajo de investigación la evaluación del método propuesto en la sección 3, mediante su aplicación en proyectos reales.

Persiguiendo este fin se utilizaron escenarios de implementación de componentes de software en distintos proyectos de desarrollo de software. Entre estos casos se evaluó la implementación de una funcionalidad de consulta de datos en una aplicación web en el contexto de un proyecto de software de una importante empresa de software en el país, que se muestra en la Sección 1, y la implementación de un componente funcional de registro de datos para una aplicación de escritorio como parte de un proyecto de la Dirección de Deportes de una casa de estudios superiores del país, mostrada en la Sección 2. Para finalizar el capítulo, se describen las conclusiones relacionadas con las actividades de instanciación del modelo situacional.

Para la evaluación del método, se hizo entrega de la definición de los fragmentos de producto y de proceso descritos en los capítulos IV y V, explicando detalladamente la notación y la terminología utilizada a los desarrolladores, solicitándoles llevaran registro de las decisiones realizadas y de los argumentos seleccionados para tomar tales de decisiones. La información que se muestra a continuación es el producto de las anotaciones realizadas durante la aplicación del método resultante y las entrevistas posteriores a la aplicación del método realizadas a los desarrolladores que implementaron los componentes de software mencionados.

El recorrido por el mapa global y los distintos mapas locales es mostrado tanto de manera gráfica como de manera tabular, dejando traza de las directivas evaluadas y los argumentos seleccionados para los dos ejemplos de instanciación documentados.

1 Funcionalidad de consulta en una aplicación web

El primero de los casos a estudiar como ejemplo de la instanciación del método situacional de Implementación de Software consiste en un componente de consultas en una aplicación web desarrollada como parte de un proyecto comercial ejecutado por una de las empresas de desarrollo de software más importantes del país.

La especificación funcional de los requisitos del sistema comprende un Documento de Análisis y un Documento de Casos de Uso. Por su parte, la especificación arquitectónica esta conformada por el Documento de Arquitectura, donde se expresa en el alto nivel las decisiones de diseño arquitectónico, así como las diferentes vistas propias de este tipo de diseño. Adicionalmente se cuenta con documentos de despliegue, diseño de Bases de Datos, descripción de campos y prototipos desarrollados para la validación de los requisitos de la aplicación. La figura 34 muestra la definición de las entradas y salidas definidas para esta instanciación del proceso de implementación.

La arquitectura de la aplicación está diseñada siguiendo patrones propios de las aplicaciones empresariales como el estilo arquitectónico de N-capas y el patrón MVC, usando tecnología Java, contando con *MyEclipse* como entorno de desarrollo de la aplicación, y utilizando marcos de trabajo de alto desempeño como Apache *Struts* 2 y *Hibernate* conectándose a un servidor de base de datos *PostgreSQL*.

El plan del proyecto, que incluye el plan de implementación, es llevado de manera formal y detallada de acuerdo a las buenas prácticas recomendadas por el *PMI* y el modelo *CMMI*.

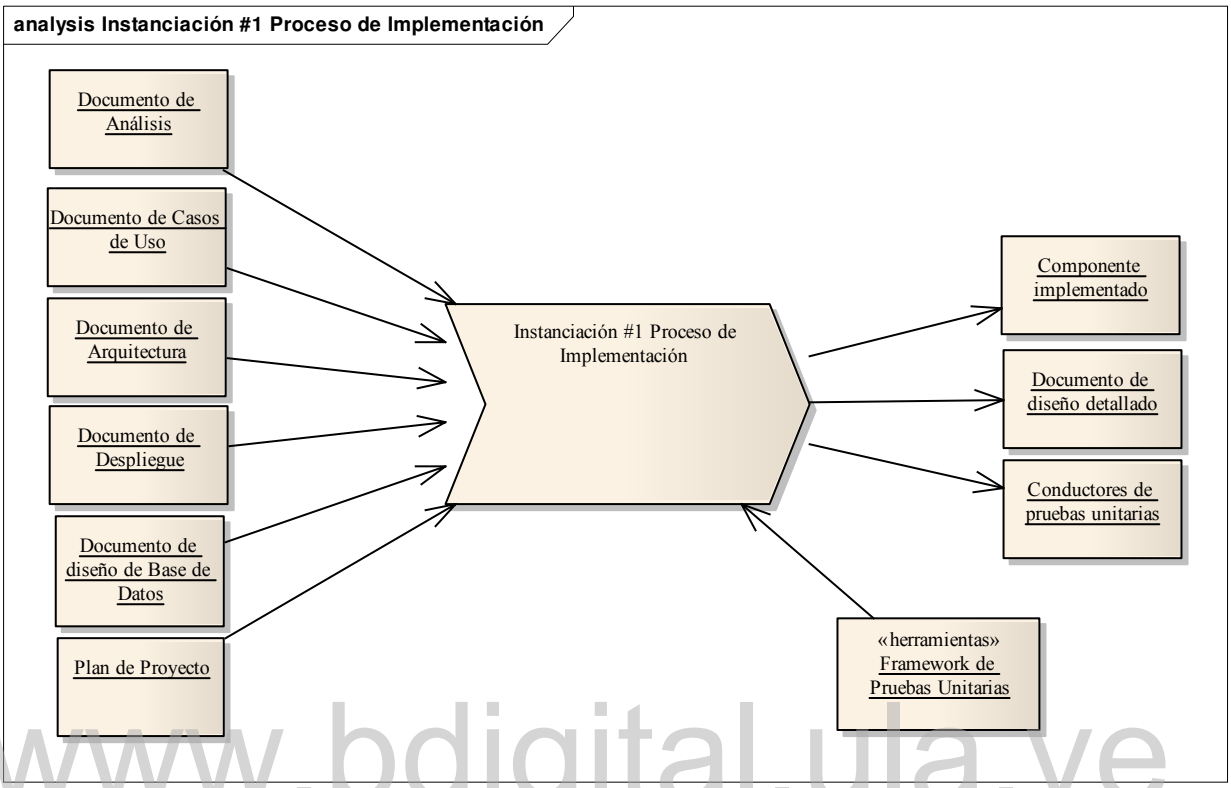


Figura 34. Entradas y salidas del Proceso de Implementación en el Escenario de Prueba #1

En cuanto a las características del personal que compone el equipo de desarrollo, está principalmente conformado por ingenieros con al menos 2 años de experiencia profesional en el desarrollo de aplicaciones, por lo que el nivel de experiencia del equipo es medio a alto, y efectivamente se plantea el enfoque del Desarrollo Guiado por Pruebas como requerido para el desarrollo de los componentes. No obstante, su nivel de conocimiento y experticia respecto al uso de herramientas automatizadas para el aumento de la productividad no es tan alto como se esperaría, por lo que hay un amplio espacio para la mejora en cuanto a las prácticas propias de este equipo de desarrollo que pueden apoyarse en tales herramientas.

El componente que se tomó como ejemplo para la aplicación del método situacional de Implementación de Software consiste en una funcionalidad de consulta, que consiste de un componente en que involucra tanto la solicitud de múltiples campos de filtrado que deben haber sido precargados desde el repositorio de datos, la ejecución asíncrona de la consulta web vía AJAX siguiendo un patrón MVC, la obtención de datos requeridos por la consulta y la generación de campos calculados a partir de las reglas de negocio establecidas para la consulta, con la correspondiente muestra de los datos resultantes en la interfaz web de usuario.

La simplicidad aparente de la funcionalidad en desarrollo se ve complementada en la asignación de desarrollo por el hecho de ser la primera funcionalidad a ser implementada en la aplicación, por lo que debe acarrear el esfuerzo asociado a la preparación de la plataforma de desarrollo y la adopción y comprobación de pertinencia de las convenciones de desarrollo acordadas por el equipo desarrollador.

1.1 Recorrido del mapa global

Luego de la asignación de las actividades de implementación para el caso de uso de Consulta de Indicadores, se procede a comenzar el recorrido del mapa global del Proceso de Implementación propuesto. Partiendo de la intención Inicio del mapa, se detecta que el Ambiente de Implementación de la aplicación no está completamente definido, por lo que debe tomarse el camino de la intención Implantar Ambiente. Los detalles de las decisiones realizadas en el contexto de esa decisión se observan más adelante en la Sección 1.2. Posteriormente se procede al aprovisionamiento del componente por lo que se procede a seguir hacia la intención Proveer Componente, cuya traza de decisiones detalladas se expone en la Sección 1.3.

Una vez que el componente ha sido aprovisionado, se encuentra listo para ser integrado a la Plataforma de ejecución de la aplicación así como al resto de componentes, procediendo por esto hacia la intención de Integrar Componente. La lista de decisiones tomadas en el contexto de esa intención se registra en la Sección 1.4. Luego de logrado el objetivo de comprobar la interoperabilidad del componente con el resto de componentes de la aplicación, se considera como culminado el Proceso de Implementación. La tabla 82 expone el detalle de los argumentos por los que se realizó el mencionado recorrido por el mapa, en tanto que la global figura 35 muestra de manera gráfica el recorrido por las intenciones del mapa global.

Tabla 82. Tabla de ejecución de mapa global en Escenario de prueba #1

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Necesidad de iniciar las actividades de implementación, aunque el Ambiente de Implementación no se encuentra preparado	Preparar el ambiente de implementación	DEI2: Implantar Ambiente por la estrategia de preparación	(A2) El *Ambiente de Implementación* está instalado pero no se ha configurado apropiadamente en lo mínimo necesario para las actividades de implementación del elemento de solución requerido.
2	Ambiente de Implementación listo para comenzar las actividades de aprovisionamiento del componente	Obtener los componentes que satisfagan los objetivos planteados en las actividades asignadas	DEI1: Proveer Componente por la estrategia de orientación a la especificación	(A1) El *Ambiente de Implementación* se encuentra configurado para el proyecto actual, y se requiere la creación o modificación en la implementación de uno o más componentes de software y (A11) El *Plan de Implementación* involucra la creación de nuevos componentes o la modificación de un componente existente por el mecanismo acordado para la gestión de cambios.
3	El componente ha sido aprovisionado mediante su codificación	Asegurar la interoperabilidad del componente con el resto de la aplicación	DEI7: Integrar Componente por la estrategia de integración en lote	(A6) La especificación de los componentes aprovisionados hasta el momento representa el insumo suficiente y necesario para ejecutar el proceso de integración de acuerdo a la estrategia acordada para ello y (A13) Los *Componentes de Software* son desarrollados por diversos equipos disjuntos con gestión separada, o las características inherentes de los componentes a integrar dificultan en gran medida la ejecución constante o

				automatizada del proceso de integración por parte de cada miembro del equipo de desarrollo, o no se tiene experiencia en prácticas de Integración Continua.
4	Se ha comprobado la interoperabilidad del componente con la plataforma y el resto de la aplicación	Finalizar el Proceso de Implementación del componente actual	DEI9: Finalizar por la estrategia de producto integrado	(A7) Todos los componentes de la especificación original para la ejecución actual del Proceso de Implementación se encuentran aprovisionados e integrados en la solución

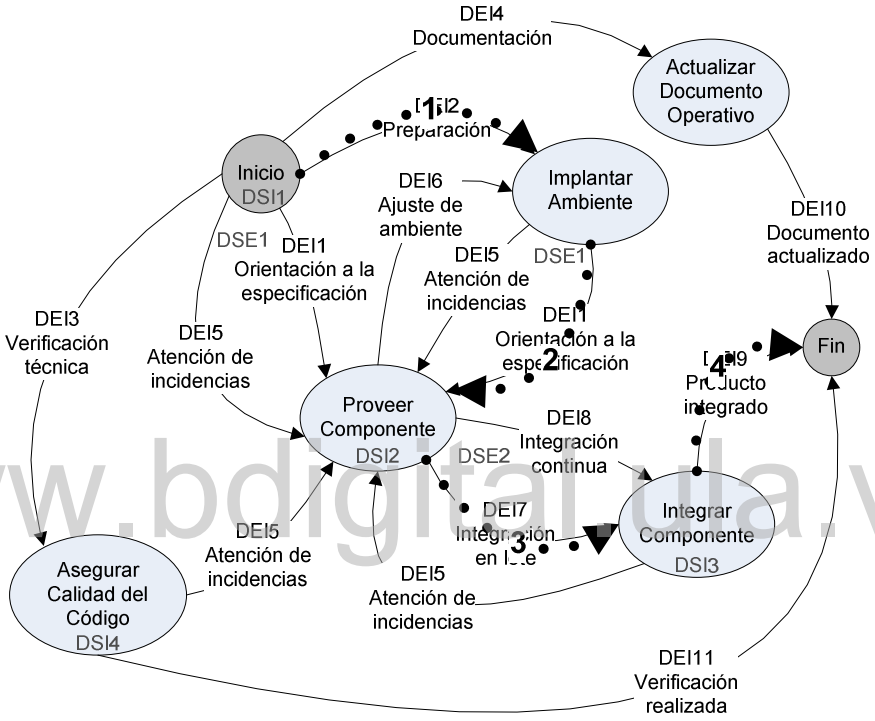


Figura 35. Recorrido de mapa global en Escenario de prueba #1

1.2 Recorrido del mapa local de Implantar Ambiente

La intención de Implantar Ambiente surge ante la situación de no contar con el *Ambiente de Implementación* al momento de iniciar las actividades de implementación del componente. En principio, la *Plataforma de Desarrollo* no estaba completamente preparada ya que faltaba configurar elementos de la misma en el computador que sería usado para la codificación del componente.

Esto conlleva a elegir la intención de Configurar Plataforma. Asimismo, el equipo de desarrollo observó que faltaba definir algunas de las *Convenciones de Desarrollo* requeridas para comenzar la construcción de los elementos de interfaz gráfica de usuario, específicamente en lo referido a los estándares de estilo de las interfaces gráficas de usuario. Por ese motivo, se continúa hacia la intención de Definir Convenciones de Desarrollo. Una vez satisfechas ambas intenciones, se procede a finalizar las actividades asociadas al mapa local de Implantar Ambiente.

La tabla 83 describe las situaciones y los argumentos esgrimidos para seleccionar las intenciones descritas antes, mientras la figura 36 muestra el recorrido gráfico sobre el mapa local de Implantar Ambiente.

Tabla 83. Tabla de ejecución de mapa local de Implantar Ambiente en Escenario de prueba #1

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Necesidad de preparar los elementos de la *Plataforma de Desarrollo* requeridos para la codificación	Configurar la *Plataforma de Desarrollo*	DEI30: Configurar Plataforma por la estrategia de configuración inicial	(A41) La plataforma de desarrollo mínima no está completamente configurada, y falta la configuración de las herramientas básicas necesarias para dar soporte al Proceso de Implementación
2	Plataforma ya configurada pero falta la definición de algunos estándares requeridos para la construcción de las interfaces gráficas de usuario	Definir *Convenciones de Desarrollo* faltantes	DEI31: Definir Convenciones de Desarrollo por la estrategia de adopción	(A42) Las convenciones de desarrollo no se encuentran completamente definidas, falta definición de estándares de construcción o de diseño de componentes
3	La plataforma y las convenciones se encuentran listas para comenzar el aprovisionamiento del componente	Continuar con las actividades de implementación del componente	DEI32: Finalizar por la estrategia de ambiente implantado	(A43) Tanto la plataforma de desarrollo como las convenciones de desarrollo se encuentran en un nivel suficiente de definición como para comenzar a realizar las actividades del Proceso de Implementación

1.3 Recorrido del mapa local de Proveer Componente

Como se mencionó anteriormente, entre las prácticas requeridas para el desarrollo del componente está el uso de la práctica de Desarrollo Guiado por Pruebas descrita en diversas ocasiones. Esto impacta la manera en que se realiza el diseño detallado y la forma en que se verifica la construcción del componente. De igual manera, para el proyecto de desarrollo se estableció que los documentos de diseño detallado forman parte de los entregables de la Solución, razón por la que se hace indispensable crearlos.

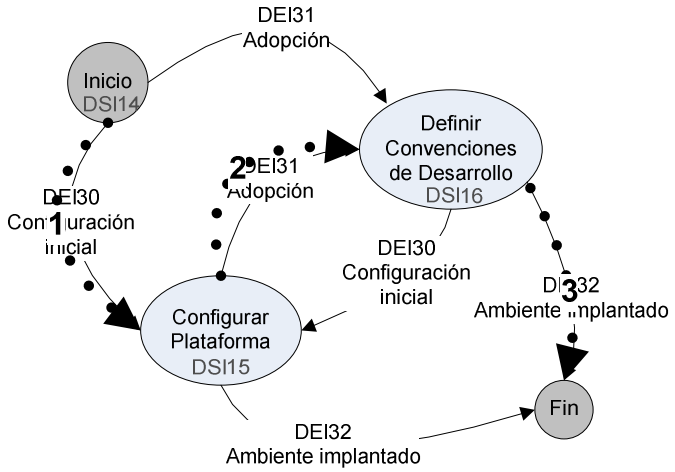


Figura 36. Recorrido de mapa local de Implantar Ambiente en Escenario de prueba #1

El aprovisionamiento de los componentes asociados al caso de uso en construcción requiere de la realización del modelado correspondiente. Por ello, una vez comenzado el recorrido por las intenciones del mapa local de Proveer Componente se selecciona la intención de Detallar Componente, donde se realiza el modelado de los componentes, el diseño de los casos de pruebas unitarias y la documentación del diseño detallado, tal como lo establecen los acuerdos realizados con el cliente. Una vez que se realizaron estas actividades, se procede a la codificación del componente, seleccionando la intención de Codificar Componente y siguiendo la estrategia de desarrollo guiado por pruebas. Una vez que se completa la codificación tanto de los *Conductores Automatizados de Pruebas Unitarias* como de los componentes en si, el proceso de verificación es realizado de inmediato al ejecutar satisfactoriamente los conductores desarrollados, una vez que se selecciona la intención de Verificar Componente. Posterior a esto, el recorrido del mapa local de Proveer Componente culmina. La tabla 84 muestra las decisiones tomadas y sus argumentos en el recorrido de este mapa local. Por su parte, la figura 37 muestra el recorrido en este mapa local de manera gráfica.

Tabla 84. Tabla de ejecución mapa local de Proveer Componente en Escenario de prueba #1

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Se cuenta con las especificaciones funcionales y arquitectónicas, pero poseen un nivel muy alto como para comenzar la codificación	Detallar y entender mejor las actividades de codificación necesarias	DEI12: Detallar Componente por la estrategia de modelado	(A15) La *Especificación* aun tiene un nivel alto y necesita ser más concreta antes de comenzar la codificación y (A29) La especificación se encuentra disponible en un formato que dificulta la generación automática del esqueleto del código o no se desea hacer uso de este tipo de facilidad.
2	Se tiene contexto de las actividades necesarias y la definición de la estructura del componente y se desea hacer uso del desarrollo guiado por pruebas	Diseñar los casos de pruebas unitarias	DEI16: Detallar Componente por la estrategia de diseño de pruebas unitarias	(A18) Aún se detectan interrogantes de implementación en el modelo detallado que deben ser respondidas antes de comenzar la codificación, por lo que se requiere un mayor nivel de granularidad en el detalle del mencionado modelo o faltan actividades que deben ser realizadas como parte del diseño detallado (por ejemplo, diseño de pruebas unitarias o la documentación del diseño detallado) y (A32) El diseño detallado ya contempla la separación de las responsabilidades entre clases, se ha completado el diseño de las interfaces del componente y se desea hacer uso del desarrollo guiado por pruebas.
3	Diseño de pruebas unitarias lista, pero se requiere realizar el documento de diseño detallado	Generar el documento de diseño detallado	DEI15: Detallar Componente por la estrategia de documentación del diseño detallado	(A18) Aún se detectan interrogantes de implementación en el modelo detallado que deben ser respondidas antes de comenzar la codificación, por lo que se requiere un mayor nivel de granularidad en el detalle del mencionado modelo o faltan actividades que deben ser realizadas como parte del diseño detallado (por ejemplo, diseño de pruebas unitarias o la documentación

				del diseño detallado) y (A31) El documento de diseño detallado es un requisito explícito del proyecto, ya sea por solicitud del cliente o por que haya sido acordado de esa manera, por estándares organizacionales. Además, la complejidad de la implementación merece una explicación más detallada que la disponible en la especificación arquitectónica. Finalmente, se requiere que el diseño esté suficientemente elaborado como para que no se invierta tiempo innecesario documentando algo que tiene alta tendencia a ser cambiado, por no haber sido comprobado aún.
4	El diseño detallado está completo, se cuenta con las pruebas unitarias y con el documento de diseño detallado	Comenzar la codificación	DEI18: Codificar Componente por la estrategia guiada por pruebas	(A20) El diseño detallado está suficientemente elaborado, las especificaciones de pruebas unitarias requeridas han sido generadas, se entienden las actividades de codificación que deben ser realizadas y se han cubierto las expectativas de documentación que se requiere durante el diseño detallado del componente y (A34) Se desea seguir la práctica de desarrollo guiado por pruebas, se posee el diseño preliminar de las pruebas unitarias a implementar, se cuenta con un *Framework* para el desarrollo de *Conductores Automatizados de Pruebas Unitarias* para el componente en desarrollo y el desarrollador tiene contexto básico del desarrollo de pruebas unitarias y sus conductores
5	Tanto los conductores de pruebas unitarias como el código de los componentes se encuentra terminado en su totalidad	Verificar que el componente ha sido construido correctamente	DEI25: Verificar Componente por la estrategia de pruebas unitarias satisfechas	(A25) Todos los elementos de trabajo asociados a la codificación han sido satisfechos y el componente ha aprobado las pruebas preliminares durante la codificación y (A37) Se ha seguido la práctica de desarrollo guiado por pruebas y las pruebas unitarias se ejecutan de manera correcta en su totalidad
6	El componente ha sido completamente codificado y verificado	Finalizar el proceso de aprovisionamiento del componente	DEI27: Finalizar por la estrategia de componente verificado	(A28) El componente ha pasado satisfactoriamente las pruebas unitarias y cumple con los requisitos de interfaz que dicta la especificación

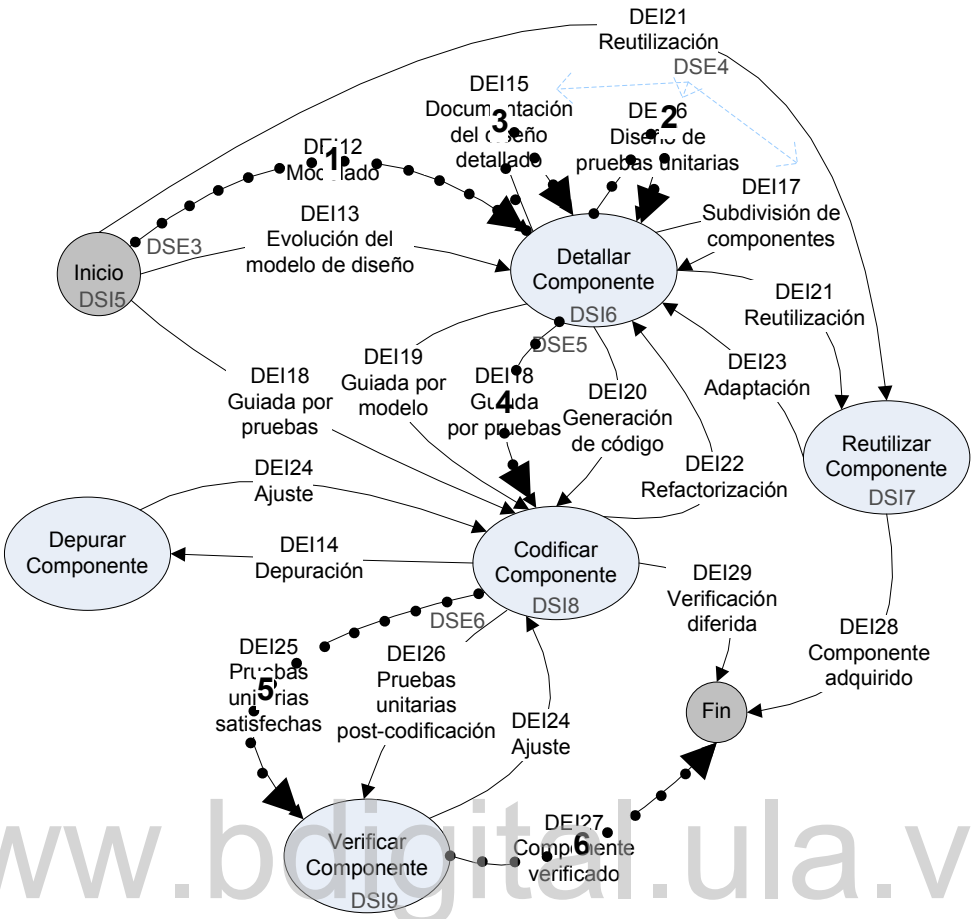


Figura 37. Recorrido de mapa local de Proveer Componente en Escenario de prueba #1

1.4 Recorrido del mapa local de Integrar Componente

A diferencia de la práctica de desarrollo guiado por pruebas que se encuentra instanciada en el equipo de trabajo del proyecto estudiado, la práctica de Integración Continua no ha sido instanciada ni es muy conocida por el equipo. Por esta razón la integración es llevada a cabo de manera manual por cada desarrollador, usando como herramienta principal el *Sistema de Control de Versiones*. La integración comienza por la implantación en la plataforma de la aplicación, verificando que efectivamente se cuenten con todas las versiones de las librerías que requiera el componente que se desarrolló. De inmediato se procede a la conexión parcial de un componente a la vez: La primera conexión se hace a nivel de la interfaz de usuario web, conectando los enlaces correspondientes con el menú de navegación del sitio web, y el ajuste de los archivos de configuración de la aplicación. Posteriormente, se procede a la verificación de la integración con la capa de datos, verificando que el componente de consultas se conecte sin problemas con el servidor de base de datos. La integración es llevada a cabo paso a paso para detectar cualquier error en la integración de manera aislada, sin involucrar todos los componentes a ser enlazados de una sola vez.

Luego de realizar las dos integraciones parciales de manera exitosa se procedió a completar el recorrido del mapa local de Integrar Componente, indicando que la integración se realizó de manera exitosa. La figura 38 muestra el recorrido en este mapa local de manera gráfica, mientras que la tabla 85 permite observar las situaciones y los argumentos observados.

Tabla 85. Tabla de ejecución de mapa local de Integrar Componente en Escenario de prueba #1

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	El componente está listo para ser integrado a la plataforma y a los componentes con los que interactúa.	Integración del componente con la plataforma de la aplicación	DEI38: Implantar en Plataforma por estrategia de implantación	(A47) La interfaz de uso del componente no es crítica para aplicaciones o servicios externos.
2	El componente se implantó en la plataforma y requiere verificación	Verificar integración parcial	DEI40: Verificar y Validar Integración Parcial por la estrategia de pruebas de integración	
3	La integración con la plataforma ha sido verificada. Está pendiente la integración con los componentes de interfaz gráfica de usuario	Enlazar el componente con los elementos de la interfaz gráfica de usuario	DEI39: Enlazar Componentes por estrategia de integración de componentes	(A49) Se ha guardado el resultado de las integraciones parciales que se han realizado hasta el momento y aun faltan componentes por ser integrados con el componente actual.
4	El componente y la interfaz gráfica de usuario están integrados. Se requiere probar la integración	Verificar integración parcial	DEI40: Verificar y Validar Integración Parcial por la estrategia de pruebas de integración	
5	La integración con la interfaz gráfica de usuario ha sido verificada. Está pendiente la integración con la capa de datos	Enlazar el componente con la capa de datos	DEI39: Enlazar Componentes por estrategia de integración de componentes	(A50) Se ha guardado el resultado de las integraciones parciales que se han realizado hasta el momento y aun faltan componentes por ser integrados con el componente actual.
6	El componente y la capa de datos están integrados. Se requiere probar la integración	Verificar integración parcial	DEI40: Verificar y Validar Integración Parcial por la estrategia de pruebas de integración	
7	Se completó la integración con todos los componentes y ningún error fue encontrado	Terminar las actividades de integración	DEI42: Finalizar por la estrategia de integración exitosa	(A51) Se han terminado los componentes por integrar o se han detectado errores que no permiten completar las pruebas en las integraciones parciales y (A52) La integración con los componentes previamente aprovisionados se realizó en su totalidad y sin arrojar error alguno

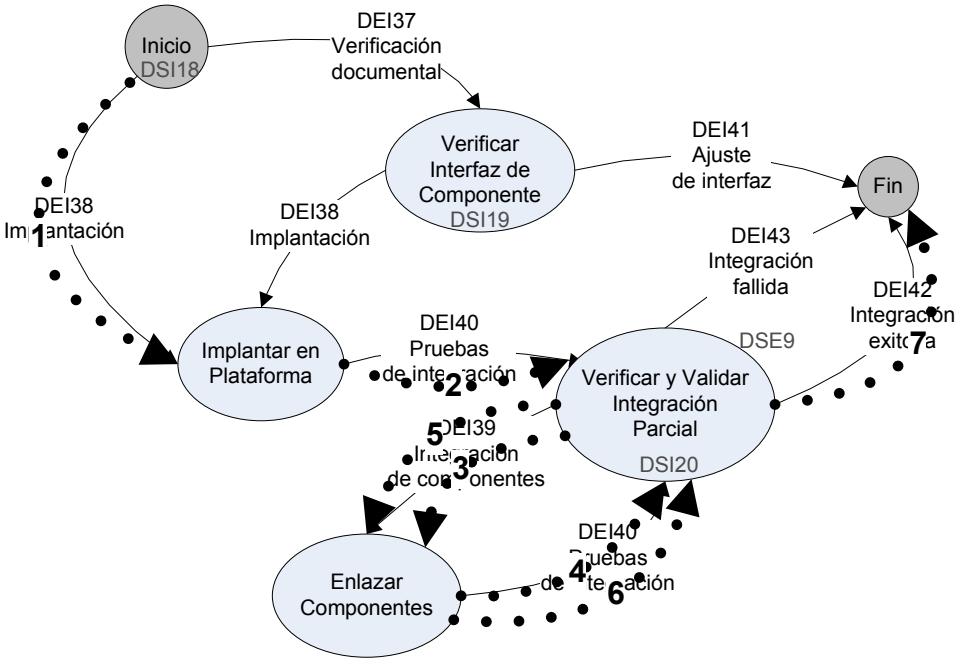


Figura 38. Recorrido de mapa local de Integrar Componente en Escenario de prueba #1

2 Componente funcional de registro de datos en aplicación de escritorio

El segundo caso usado para comprobar la instanciación del Proceso de Implementación desarrollado tuvo lugar en un proyecto interno de desarrollo de software llevado a cabo en la Dirección de Deportes de una casa de estudios en el país. Este proyecto está siendo desarrollado como pasantías profesionales por estudiantes de la carrera Ingeniería en Informática dictada en una de las facultades de la misma casa de estudios, evidenciando esto que el equipo de desarrollo está formado principalmente por personas con poca experiencia en el desarrollo profesional de aplicaciones, y poco conocimiento acerca de técnicas ágiles y de herramientas que permitan aumentar su productividad.

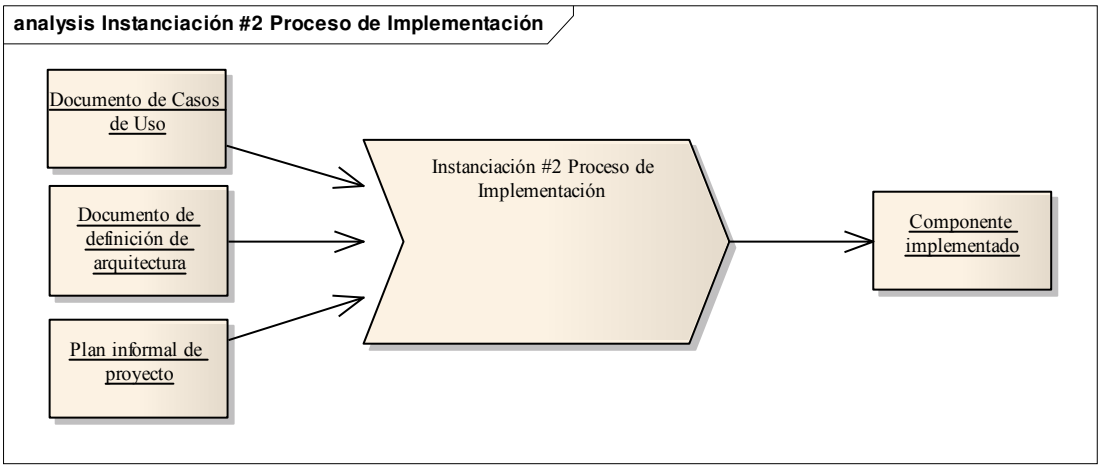


Figura 39. Entradas y salidas del Proceso de Implementación en el Escenario de prueba #2

La gestión de los requisitos en el mencionado proyecto se realiza principalmente a través de la documentación de casos de uso, conteniéndose allí la especificación funcional de la aplicación. La arquitectura de la aplicación está escrita en el Documento de Definición de Arquitectura y contempla que la aplicación debe ser implementada como una aplicación de escritorio con conexión a un servidor de datos remoto usando como lenguaje de programación Java. Por su parte, el plan del proyecto, incluyendo el plan de implementación, es llevado de manera poco formal por el tutor industrial de las mencionadas pasantías. La figura 39 muestra de manera gráfica las entradas y salidas del Proceso de Implementación para la instanciación actual.

El proyecto define su Plataforma de Desarrollo usando el entorno de desarrollo Eclipse y el editor gráfico del marco de trabajo para interfaces gráficas ZK. Entre las *Convenciones de Desarrollo* se contempla el estándar de estilo de la interfaz de usuario, el estándar de nomenclatura de los componentes, el estándar de nomenclatura de la base de datos, entre otros.

El componente específico está inscrito en el Caso de Uso de Registro de Consulta Médica de Atletas, el cual es uno de los casos de usos centrales de la aplicación en desarrollo. Dicho componente tuvo durante su desarrollo amplias expectativas de ser reutilizado, por cuanto existen otros casos de uso con similar estructura e importancia para el proyecto.

2.1 Recorrido del mapa global

Partiendo de la intención de inicio, se verifica que el *Ambiente de Implementación* se encuentra completamente definido y preparado, por lo que se procede directamente al aprovisionamiento del componente, guiado por las especificaciones indicadas junto a la asignación de las actividades de implementación.

Una vez que el componente ha sido aprovisionado se procede con la integración. Aquí se evidencia la poca experiencia del equipo y lo rudimentario de la técnica usada para la integración, por cuanto consiste en la copia y sobreescritura de los archivos de código fuente entre el equipo usado para el desarrollo y el equipo usado para la integración de los diversos componentes implementados. Una vez completada la integración, se considera que la ejecución del proceso esbozado en el mapa global de método situacional se da por concluida.

La tabla 86 muestra las situaciones y argumentos usados para seleccionar los caminos sobre el mapa global, mientras que la figura 40 muestra gráficamente el recorrido por el mencionado mapa.

Tabla 86. Tabla de ejecución de mapa global en Escenario de prueba #2

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Necesidad de iniciar las actividades de implementación	Construir el componente requerido por la actividad asignada	DEI1: Proveer Componente por la estrategia de orientación a la especificación	(A1) El *Ambiente de Implementación* se encuentra configurado para el proyecto actual, y se requiere la creación o modificación en la implementación de uno o más componentes de software y (A11) El *Plan de Implementación* involucra la creación de nuevos componentes o la modificación de un componente existente por el mecanismo acordado para la gestión de cambios.

2	El componente solicitado ya ha sido aprovisionado. Se requiere realizar la integración del componente al resto de los elementos de la Solución	Integrar el componente aprovisionado a la aplicación	DEI7: Integrar Componente por la estrategia de integración en lote	(A6) La especificación de los componentes aprovisionados hasta el momento representa el insumo suficiente y necesario para ejecutar el proceso de integración de acuerdo a la estrategia acordada para ello y (A13) Los *Componentes de Software* son desarrollados por diversos equipos disjuntos con gestión separada, o las características inherentes de los componentes a integrar dificultan en gran medida la ejecución constante o automatizada del proceso de integración por parte de cada miembro del equipo de desarrollo, o no se tiene experiencia en prácticas de Integración Continua.
3	El componente se encuentra aprovisionado e integrado a la *Solución*	Finalizar el Proceso de Implementación para el componente asignado	DEI9: Finalizar por la estrategia de producto integrado	(A7) Todos los componentes de la especificación original para la ejecución actual del Proceso de Implementación se encuentran aprovisionados e integrados en la solución

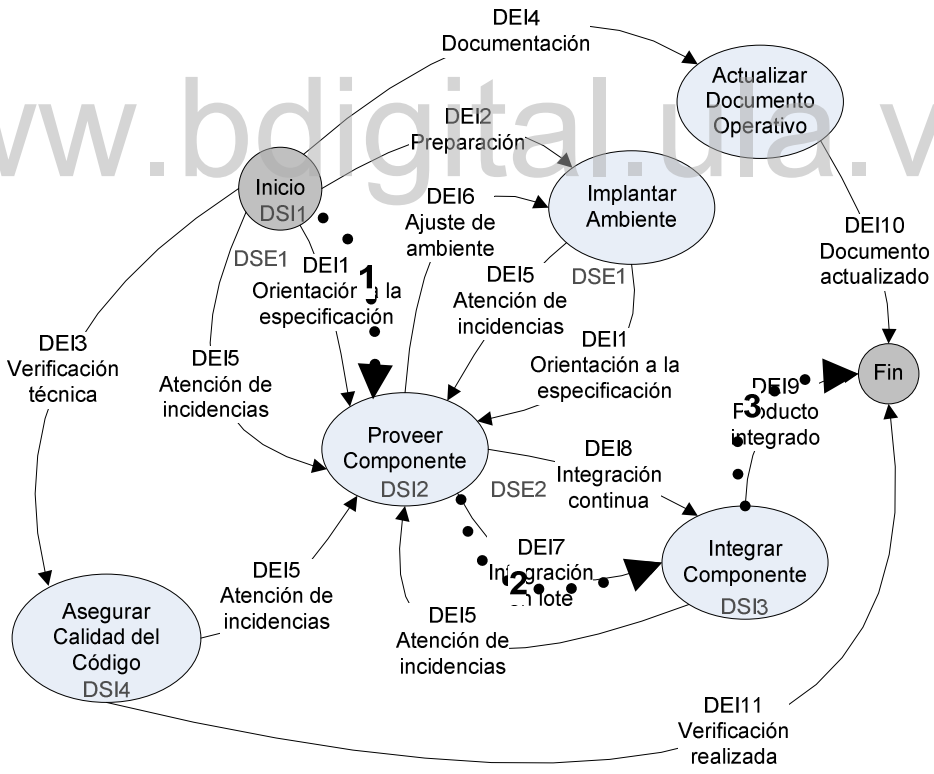


Figura 40. Recorrido de mapa global en Escenario de prueba #2

2.2 Recorrido del mapa local: Proveer Componente

Para el aprovisionamiento del componente se parte de la intención de Inicio del mapa local de Proveer Componente y se comienza por la necesidad de concretar las actividades de implementación a partir de las

especificaciones funcionales y arquitectónicas, lo cual se lleva a cabo en la práctica de modelado, siguiendo la intención de Detallar Componente. Esta actividad es llevada a cabo por el equipo de desarrollo de este proyecto partiendo del diseño de la interfaz gráfica de usuario, y entendiendo las necesidades que puedan tener los actores al avanzar en la ejecución del caso de uso relacionado. Dado que el desarrollo del componente involucra incluso el diseño de esa interfaz de usuario, el proceso de modelado es realizado de manera intermitente con actividades de diseño y construcción de esa interfaz. Esta práctica puede tener sus bemoles y contras, pero es usada con frecuencia, especialmente con técnicas de prototipos y desarrollo rápido de aplicaciones (RAD).

Una de las prácticas también asociadas al modelado por parte de este equipo de desarrolladores es la búsqueda de similitudes entre el componente que se desarrolla actualmente y alguno de los componentes desarrollados anteriormente, con el fin de detectar oportunidades de reutilización, sea ya de un componente reutilizable completo o de alguna estrategia, o tan sólo para aprender rápidamente como llevar a cabo una tareas específica. Efectivamente en el desarrollo del componente asignado, se detectó la oportunidad de reutilizar funciones que habían sido usadas en componentes anteriores, por lo que se procedió a evaluar la reutilización del componente con la intención de Reutilizar Componente. Las funciones y métodos que serían reutilizados requerían la adaptación previa antes de poder ser llamados desde el código del nuevo componente, por lo que se procedió al modelado de la adaptación específica.

La adaptación de los componentes de software requería un ajuste en su diseño para evitar duplicación de código. Por la poca formalidad y ceremonia del proceso usado en este proyecto, el proyecto no exige la creación de documentos de diseño detallado o casos de pruebas unitarias como parte del detallado del componente. Una vez modelada la adaptación y extraídos una lista de elementos a ser implementados durante la codificación, se procede a realizar esa siguiente actividad en el subproceso de Proveer Componente. La codificación es realizada siguiendo la estrategia guiada por el modelo resultante del diseño detallado, dado que no se cuenta con la experticia para la aplicación de prácticas de desarrollo guiado por pruebas. En su lugar, el equipo de desarrollo sigue la secuencia de eventos de interfaz de usuario, como el hacer clic en un botón o el presionar la tecla *Enter* ante una entrada de datos, y es implementada pasando por los componentes del patrón MVC, por las clases de lógica de la aplicación y por las distintas consultas de base de datos, y así mismo de regreso por las distintas capas hasta que se muestra el resultado visible de la funcionalidad implementada. Este mismo patrón se aplica para cada distinto evento de interfaz hasta que se culminan todas las funcionalidades asociadas al caso de uso asignado.

En medio de la implementación de las funcionalidades, el equipo se topó con un error que no permitía el registro de los datos de la consulta, donde fue oportuno dirigirse a la intención de Depurar Componente. Aquí, el equipo de desarrollo uso diversas técnicas como el envío de mensajes por consola, el registro de errores a través de un archivo de log y el uso de puntos de interrupción y depurador interactivo. De esta manera pudo encontrarse errores asociados a la capa de datos, lo cual se resolvió realizando ajustes en la codificación.

Posteriormente a la codificación, para realizar las verificaciones asociadas al componente se requirió la ejecución de pruebas unitarias post-codificación. Estas pruebas unitarias fueron realizadas principalmente de manera manual y haciendo uso de conductores vía consola, sin usar un *Framework* de pruebas automatizadas. No obstante, la verificación de la correctitud de la construcción del componente fue realizada y de esa manera culminó el subproceso de aprovisionamiento del componente. La tabla 87 muestra un resumen de las situaciones, directivas realizadas y los argumentos que soportaron las decisiones realizadas durante el recorrido del mapa local, mientras que la figura 41 muestra el recorrido por el mapa local.

Tabla 87. Tabla de ejecución de mapa local de Proveer Componente en Escenario de prueba #2

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Especificación funcional y arquitectónica con nivel muy alto, poco concreto	Modelar las actividades de implementación	DEI12: Detallar Componente por la estrategia de modelado	(A15) La *Especificación* aun tiene un nivel alto y necesita ser más concreta antes de comenzar la codificación y (A29) La especificación se encuentra disponible en un formato que dificulta la generación automática del esqueleto del código o no se desea hacer uso de este tipo de facilidad.
2	Una parte del componente a implementar puede ser reutilizada desde otro componente	Reutilizar porciones de componentes anteriormente desarrollados	DEI21: Reutilizar Componente por la estrategia de reutilización	(A19) Se ha detectado que la funcionalidad requerida para el componente que se está detallando puede ser cubierta de manera parcial o total por un componente disponible en alguna de las bibliotecas de componentes reutilizables del proyecto
3	El componente a ser reutilizado requiere una modificación previa	Adaptar componente a reutilizar para su aplicación	DEI23: Detallar Componente por la estrategia de adaptación	(A21) El componente reutilizable requiere una adaptación antes de que pueda ser utilizado en la solución en desarrollo
4	El diseño se encuentra suficientemente concreto para comenzar la codificación	Comenzar la codificación del componente y la modificación del componente a reutilizar	DEI19: Codificar Componente por la estrategia guiada por modelo	(A20) El diseño detallado está suficientemente elaborado, las especificaciones de pruebas unitarias requeridas han sido generadas, se entienden las actividades de codificación que deben ser realizadas y se han cubierto las expectativas de documentación que se requiere durante el diseño detallado del componente y (A35) No se dispone de alguno de los requisitos del desarrollo guiado por pruebas o con la capacidad de generar código a partir del diseño detallado digitalizado del componente
5	Se ha detectado un error en el código y su resolución no es evidente	Depurar código del componente	DEI14: Depurar Componente por la estrategia de depuración	(A24) Se ha detectado un comportamiento no esperado durante las pruebas iniciales del componente y no es evidente la razón del mismo
6	El error ha sido encontrado y se requiere al respectivo ajuste en el código	Ajustar código del componente	DEI24: Codificar Componente por la estrategia de ajuste	
7	La codificación ha concluido y se requiere verificar la correcta codificación de la funcionalidad	Verificar funcionamiento del componente	DEI26: Verificar Componente por la estrategia de pruebas unitarias post-codificación	(A25) Todos los elementos de trabajo asociados a la codificación han sido satisfechos y el componente ha aprobado las pruebas preliminares durante la codificación y (A38) No se dispone de pruebas

				unitarias para el componente actual, sin embargo ya las actividades de codificación se concluyeron
8	El componente ha sido completamente codificado y verificado	Finalizar el proceso de aprovisionamiento del componente	DEI27: Finalizar por la estrategia de componente verificado	(A28) El componente ha pasado satisfactoriamente las pruebas unitarias y cumple con los requisitos de interfaz que dicta la especificación

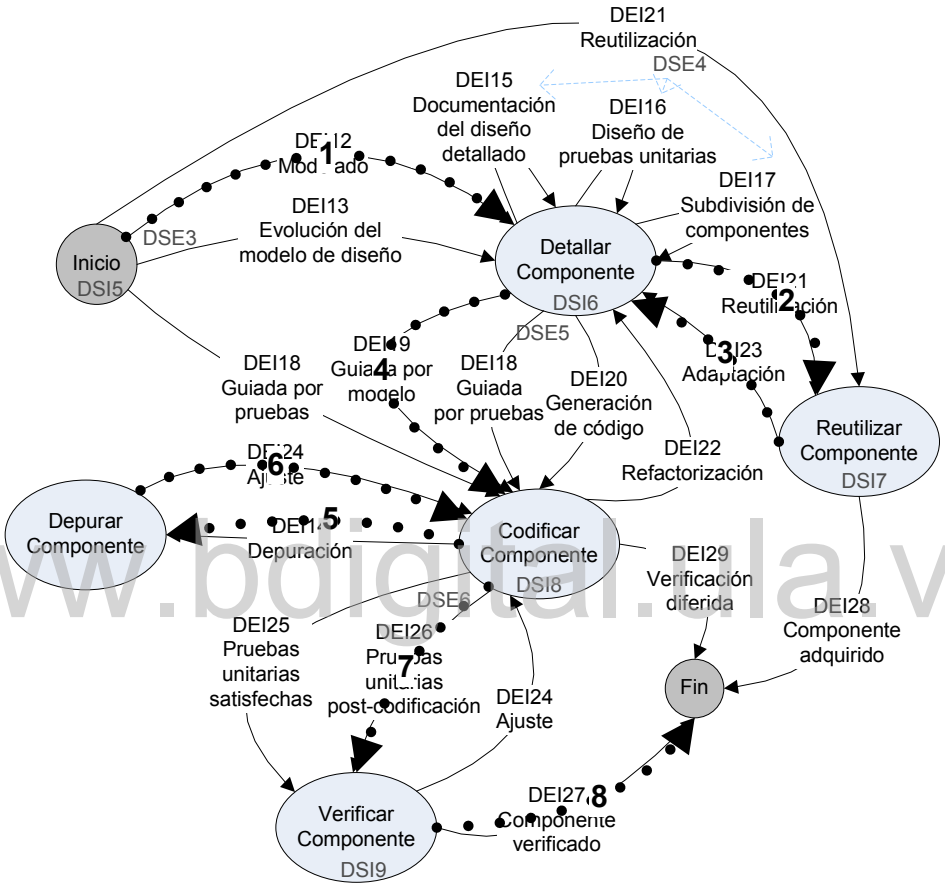


Figura 41. Recorrido de mapa local de Proveer Componente en Escenario de prueba #2

2.3 Recorrido del mapa local: Integrar Componente

El proceso de integración del componente aprovisionado, como se mencionó en el recorrido del mapa global, sigue una técnica algo rudimentaria, lo cual evidencia la inexperiencia del equipo de desarrollo del proyecto en estudio. Esta técnica es principalmente la sobreescritura de los archivos de código fuente entre los equipos de desarrollo y el equipo utilizado para la integración. Debido a esta técnica, la integración es realizada de manera total, es decir, al implantar el nuevo componente en la plataforma también se está enlazando a los otros componentes con los que interactúa, por lo que es una práctica que no se recomienda.

Desde la intención de Inicio se procede a la implantación en la Plataforma de ejecución de la *Solución*, mediante la sobreescritura de los archivos fuentes directamente en el equipo de integración. De inmediato se procede con las pruebas de integración que se desarrollan de manera similar a los conductores de pruebas unitarias usados, mediante pruebas más bien globales y que buscan verificar la funcionalidad del componente

(Pruebas Funcionales). Una vez que se ha verificado el funcionamiento del componente, se da por exitosa la integración y se culmina de esta manera el subproceso de integración de componentes. La tabla 88 muestra el detalle de las decisiones realizadas, mientras la figura 42 muestra el recorrido de manera gráfica.

Tabla 88. Tabla de ejecución de mapa local de Integrar Componente en Escenario de prueba #2

#	Situación Inicial	Objetivo	Directiva de Ejecución de Intención realizada	Argumento
1	Componente listo para ser integrado	Integrar el nuevo componente al resto de la aplicación	DEI38: Implantar en Plataforma por estrategia de implantación	(A47) La interfaz de uso del componente no es crítica para aplicaciones o servicios externos.
2	El nuevo componente y el resto de la aplicación están integrados	Verificar integración	DEI40: Verificar y Validar Integración Parcial por la estrategia de pruebas de integración	
3	Se completó la integración con todos los componentes y ningún error fue encontrado	Terminar las actividades de integración	DEI42: Finalizar por la estrategia de integración exitosa	(A51) Se han terminado los componentes por integrar o se han detectado errores que no permiten completar las pruebas en las integraciones parciales y (A52) La integración con los componentes previamente aprovisionados se realizó en su totalidad y sin arrojar error alguno

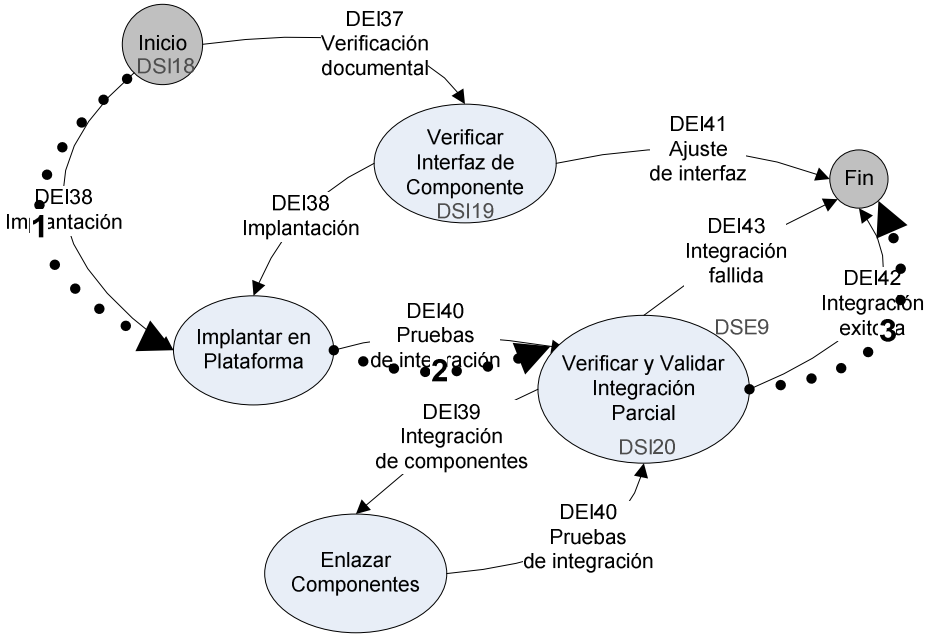


Figura 42. Recorrido de mapa local de Integrar Componente en Escenario de prueba #2

3 Conclusiones de este capítulo

Luego de haber aplicado en dos ocasiones el método situacional propuesto para el Proceso de Implementación se han podido obtener interesantes reflexiones acerca del uso que recibió la misma especificación del método por parte de quienes lo probaron:

- En el caso del Escenario #2, donde el equipo de desarrollo tenía poca o ninguna experiencia con prácticas o herramientas que potenciaban la agilidad del desarrollo y su productividad, se pudo conocer que la especificación de los fragmentos situacionales así como los argumentos expuestos para su elección y aplicación de acuerdo a la ocasión, resultaron ser útiles para el aprendizaje y el conocimiento de la existencia de tales prácticas y herramientas. Efectivamente el equipo de desarrollo expresó su intención de prepararse en el uso y funcionamiento de las herramientas comentadas como parte de los fragmentos situacionales, de la misma manera que expresó su interés en continuar aprendiendo sobre las técnicas y prácticas que son soportadas por las mismas herramientas.

- Los equipos de desarrollo que aplicaron el método expuesto comentaron que les resultó de gran utilidad el conocer los argumentos por los que debía seguirse un paso u otro. En conjunto con la conclusión anterior, esto nos sirve de argumento para inferir que la representación situacional de las prácticas colabora en la gestión del conocimiento y su difusión entre los miembros del equipo de desarrollo.

- Un importante punto de mejora comentado por los desarrolladores que colaboraron en la instanciación del método situacional está en la representación estática documental del método, lo cual exige de ellos seguir la referencia entre los distintos mapas y las tablas de las directivas para mantener el hilo de ejecución del método. La propuesta que se hace en este sentido es la de la implementación de una herramienta que facilite la lectura de las distintas intenciones, estrategias y argumentos para su aplicación, así como el salto entre referencias (hiperenlaces) de los mismos elementos.

CAPITULO VIII: Conclusiones y recomendaciones

1 Conclusiones

En el transcurso del desarrollo de este trabajo surgieron aspectos de gran relevancia deben ser destacados como conclusiones del mismo

- Se evaluaron distintos modelos de implementación de los enfoques ágil y disciplinado para extraer las similitudes y diferencias entre ambos enfoques, y de esta evaluación se pudo crear una serie de lineamientos sobre los cuales es factible definir los principios sobre los cuales puede construirse métodos que contemplen características ágiles y disciplinadas.

- Se definieron un Modelo de Productos y un Modelo de Procesos que contienen las definiciones de los fragmentos situacionales de producto y proceso, respectivamente, que serán de utilidad para la construcción de métodos situacionales adaptables a las circunstancias de cada proyecto.

- Se aplicó el método situacional resultante en la implementación de componentes en dos proyectos distintos, comprobando así su capacidad para adaptarse a distintas situaciones y permitiendo observar su funcionamiento en circunstancias distintas.

- El método situacional aporta gran valor para introducir a equipos de desarrollo con poca o ninguna experiencia en el uso de prácticas y herramientas que promueven la agilidad e impulsan la productividad durante las actividades de implementación.

- La notación situacional de los fragmentos de métodos tiene gran valor para la definición modular de prácticas y componentes de métodos que pueden ser ensamblados de acuerdo a las situaciones propias de cada proyecto.

- A pesar de las fortalezas mencionadas en las conclusiones previas, esa misma notación situacional involucra la inclusión de una complejidad inherente a la lectura y navegación a través de los múltiples saltos y referencias cruzadas entre intenciones, estrategias, directivas, argumentos y contextos. La representación estática de este tipo de notación es de difícil lectura y ello inhibe la fácil adopción de esta notación. La construcción de herramientas que ayuden a superar esta barrera de entrada tendrán un gran impacto en la comprensión de esta manera de representar métodos y maneras de hacer las cosas.

- La definición de los argumentos en un lenguaje formal y que evalúe expresiones basadas en la situación y las partes de producto del Modelo de Productos permitiría la construcción de un motor de toma de decisión para una adaptación preliminar de un método situacional a una caracterización paramétrica de un proyecto. Con la definición de esta representación formal, es factible generar una representación no situacional dado un conjunto de decisiones tomadas a priori que puede ser en forma algorítmica o incluso siguiendo un diagrama de flujos, lo cual mejoraría enormemente la legibilidad de los métodos situacionales una vez que no hay que tomar decisión alguna.

- Como parte de los resultados de la investigación asociada al Capítulo II sobre el estado del arte de modelos de Procesos de Implementación, se cuenta con un artículo en la prestigiosa revista Ciencia & Ingeniería de la Facultad de Ingeniería de la Universidad de Los Andes.

- Las descripciones y representaciones de métodos, en especial en el ámbito del desarrollo de software, no solo cumplen una función orientadora ante la duda y una función institucional de implantación de las mejores prácticas, sino que también cumplen un rol importante como insumo para la discusión y la evaluación del estado actual de las prácticas que promueva la evolución y mejora en el tiempo, especialmente si permite indicar el por qué se persigue la consecución de un objetivo de una manera específica y se proveen adicionalmente mecanismos para la captura de la retroalimentación en el uso y aplicación de esos métodos. La representación situacional de los fragmentos de método aportan gran valor en este sentido gracias a la descripción explícita de los argumentos por los cuales debe seleccionarse una estrategia u otra para lograr un objetivo.

- Como conclusión final, pero no menos importante, se tiene que este trabajo demuestra que es factible la integración de prácticas ágiles y tradicionales en un método ágil y disciplinado sin hacer una elección estática de qué prácticas deben ser llevadas a cabo en todos los casos, sino que se puede mantener una base de fragmentos de las prácticas de ambos enfoques para su ensamblaje y elección de acuerdo a las circunstancias de cada proyecto.

2 Recomendaciones

En vista de la cantidad de elementos que no se hicieron como parte de esta tesis por encontrarse fuera del alcance de ella, se recomienda como trabajo futuro lo siguiente:

- Completar la definición de los contextos plan, en especial, la realización de los grafos de precedencia que indican el flujo de ejecución de cada uno de ellos, así como su descripción en notación formal.

- Comprobar la estrategia usada en esta tesis con otros procesos pertenecientes al desarrollo de software, como Ingeniería de Requisitos, Diseño Arquitectónico, Verificación y Validación, Gestión de Proyectos, entre otros, para lograr levantar los fragmentos situacionales que puedan ser compatibles con el Proceso de Implementación aquí definido.

- Crear una herramienta que permita la fácil lectura, edición, actualización y creación colectiva de nuevas intenciones, estrategias, contextos y argumentos para sacar provecho de la inteligencia colectiva y crear bases de fragmentos que sean soportadas por comunidades de ingenieros de software e ingenieros de métodos para el beneficio de toda la comunidad.

- Promover el diseño e implementación un motor de evaluación de argumentos y situaciones escritos en lenguaje formal para la evaluación automatizada y la recuperación de los fragmentos de método situacional de una base de métodos.

Apéndice A: Glosario de Términos

Este glosario procura la utilización del Glosario de Términos Estándar para Ingeniería de Software de la *IEEE* (*IEEE*-610.12-1990), salvo en aquellos casos en los que la definición estándar dificulta la inclusión o la integración de conceptos de áreas nuevas, como lo es el desarrollo ágil. En tales casos, se hará una aclaratoria explícita.

- ***Ambiente de Implementación:*** Conjunto de los recursos físicos tangibles e intangibles que precondicionan la implementación de componentes durante un proyecto de desarrollo de software, y que no son indispensables para las fases anteriores. Esto incluye la Plataforma y las Convenciones de Desarrollo.

- ***Base de Datos de Incidencias:*** Repositorio de datos que será usado para registrar y consultar errores, incidencias y no conformidades a lo largo de un proyecto de desarrollo de software. Puede ser tan informal como una lista de correos o una hoja de cálculo, o tan especializado como una aplicación distribuida que registra y notifica acerca de la ocurrencia o actualización de una incidencia.

- ***Biblioteca de Componentes Reutilizables:*** Una colección controlada de Componentes Reutilizables y documentación relacionada diseñado para ser usado en el uso, desarrollo y mantenimiento de software. Este tipo de colección generalmente cuenta con facilidades para buscar componentes que satisfagan una serie de criterios.

- ***Casos de Pruebas:*** Un conjunto de entradas de prueba, condiciones de ejecución y salidas esperadas, desarrollado con el objeto principal de ejercitar un programa particular o verificar su adhesión a un requisito específico.

- ***Código Fuente:*** Un conjunto de instrucciones de computadora y definiciones de datos expresados de una manera procesable como entrada por un ensamblador, compilador u otro traductor de código.

- ***Código Objeto:*** Un conjunto de instrucciones de computadora y definiciones de datos en una formato generado como salida de un ensamblador o compilador.

- ***Componente de Base de Datos:*** Componente de un sistema software que tiene como finalidad principal el almacenamiento, resguardo y la recuperación de grandes cantidades de datos, para su uso de manera persistente en el tiempo.

- ***Componente de Interfaz de Usuario:*** Componente de un sistema software que tiene como finalidad principal permitir la interacción entre una máquina y un usuario humano.

- ***Componente de Lógica de Dominio:*** Componente de un sistema software que tiene un papel activo en las aplicaciones de computadora, y que son los principales contenedores del conocimiento acerca del dominio de la aplicación y de las reglas de negocio automatizadas en una aplicación.

- ***Componente de Software:*** Una de las partes que forman un sistema software. Otros términos similares son "*Módulo*" y "*Unidad*".

- ***Componente Reutilizable:*** Se refiere a un componente de software que puede ser usado en más de un programa de computadora o sistema software. Generalmente está formado por una interfaz de componente, que define la forma en que interactúa con otros componentes o con los sistemas software, y por una implementación de componente, que consiste en los recursos internos usados por el componente para proveer los servicios publicados a través de su interfaz de uso.

- ***Conductor de Pruebas:*** Un módulo de software usado para invocar otro módulo bajo prueba y, frecuentemente, provee entradas de prueba, controla y monitorea la ejecución y reporta los resultados de las pruebas.

- ***Convenciones de Desarrollo:*** Conjunto de acuerdos previos a la implementación de un sistema software. Superconjunto que suma los estándares de construcción con la definición de las bibliotecas de componentes reutilizables a ser usadas durante el desarrollo de software.

- ***Diseño Detallado:*** El proceso de refinar y expander el diseño preliminar de un sistema o componente hasta el punto en que el diseño es suficientemente completo para ser implementado.

- ***Documento Operativo:*** Un documento que presenta la información necesaria para emplear, mantener o instalar un sistema o componente.

- ***Elemento de Solución:*** Cada uno de las partes integrantes de una solución que son consideradas como productos finales de la solución.

- ***Elemento de Trabajo:*** Unidad básica de asignación de trabajo de implementación de un componente a un desarrollador específico. Consiste en una tarea lo suficientemente simple como para poder ser estimada y realizada sin requerir una planificación formal previa.

- ***Especificación:*** Un modelo o documento que especifica, de una manera completa, precisa y verificable, los requisitos, diseño, comportamiento u otras características de un sistema o componente y, frecuentemente, los procedimientos para determinar si tales indicaciones han sido satisfechas.

- ***Especificación Arquitectónica:*** Un modelo o documento que especifica, de manera completa, precisa y verificable, la estructura organizacional de un sistema o componente.

- ***Especificación de Requisitos:*** Documentación de los requisitos esenciales (funciones, desempeño, restricciones de diseño y atributos) del software y sus interfaces externas.

- ***Estándar de Construcción:*** Requerimientos obligatorios empleados para prescribir un enfoque uniforme y disciplinado para el desarrollo de software.

- ***Framework de Pruebas Automatizadas:*** Conjunto de aplicaciones, herramientas y librerías que permiten la rápida creación y ejecución de pruebas de componentes, métodos, clases y subsistemas, directamente conectándose a los componentes a ser probados. Este tipo de pruebas no requiere acceder a las aplicaciones a través de la interfaz de usuario.

- ***Guión de Generación de Versiones Ejecutables:*** Llamado también *script,* es un programa en un lenguaje dinámico que permite automatizar el proceso de compilación e integración básica de los componentes de software, y alistarlo para la ejecución del mismo.

- ***Implementación de Software:*** Proceso y resultado de traducir un diseño en los componentes de hardware, software o ambos.

- ***Incidencia:*** Error o no conformidad detectada en relación con la especificación funcional o arquitectónica, convenciones de desarrollo establecidas o defecto en la ejecución de alguna funcionalidad de la solución.

- ***Interfaz de Componente:*** Porción de un componente reutilizable que permite la conexión con otros componentes con el fin de pasar información entre uno y otro.

- ***Modelo de Diseño Detallado:*** Modelo resultante del proceso de diseño detallado, que será usado para guiar las actividades de construcción de software.

- ***Plan de Implementación:*** Documento subconjunto del Plan del Proyecto que describe las actividades técnicas a ser realizadas durante la Implementación del Software. Por las características propias de esta fase del proyecto, las actividades están relacionadas con la construcción de los componentes de software. Describe los componentes a realizar, los recursos, los hitos a ser alcanzados y la manera en que el proyecto será organizado.

- ***Plan de Verificación y Validación:*** Documento subconjunto del Plan del Proyecto que describe las actividades de Verificación y Validación del producto a lo largo de un proyecto de desarrollo de software. Contempla las actividades, recursos, método, procedimientos e hitos a ser alcanzados asociados a asegurar la calidad del producto en construcción.

- ***Plan del Proyecto:*** Un documento que describe las actividades técnicas y administrativas a ser llevadas a cabo por un proyecto. Típicamente describe el trabajo que debe ser realizado, los recursos requeridos, los métodos a ser usados, los procedimientos a ser seguidos, los hitos a ser alcanzados y la manera en que el proyecto será organizado.

- ***Plataforma de Desarrollo:*** Conjunto de aplicaciones, herramientas, sistemas operativos, librerías y programas utilitarios que deben estar disponibles para la construcción y despliegue de los componentes de software de la solución.

- ***Producto de Software:*** Conjunto completo de programas de computadora que forman parte de una Solución y que han sido designados para ser entregados a los usuarios o clientes.

- ***Programa Fuente:*** Un programa de computadora que debe ser compilado, ensamblado o de alguna manera traducido para poder ser ejecutado por un computador.

- ***Programa Objeto:*** Un programa de computadora que es la salida resultante de un ensamblador o compilador.

- ***Pruebas de Integración:*** Actividad de pruebas en la cual los componentes de software, componentes de hardware o ambos son combinados y probados para evaluar la interacción entre ellos.

- ***Pruebas Unitarias:*** Actividad de pruebas de unidades de hardware o software o grupos de unidades relacionadas.

- ***Secuencia de Integración:*** Orden específico determinado en que debe ser integrado un conjunto específico de componentes.

- ***Servidor de Integración Continua:*** Herramienta de software que ayuda en la instanciación de la práctica de Integración Continua. Con ella se puede planificar a intervalos de tiempo regulares la obtención del último código fuente disponible desde el Sistema de Control de Versiones, su compilación y integración para la ejecución de pruebas automatizadas y de cualquier otro tipo de herramienta que, ejecutada automáticamente, permita conocer el estado actual del código y la notificación de errores al equipo de desarrollo de manera oportuna.

- ***Sistema de Control de Versiones:*** Herramienta de software que permite llevar control y registro de los cambios que se realizan sobre el código fuente de una aplicación de software.

- ***Solución:*** Conjunto de todos los productos finales que serán entregados al cliente a lo largo de todo el proyecto, incluyendo esto el Producto de Software, la Documentación Operativa del producto y cualquier otro producto de trabajo que haya sido acordado entre los interesados y el equipo de desarrollo.

- ***Validación:*** Proceso de evaluar de un sistema o componente durante o al final del proceso de desarrollo para determinar si satisface requisitos específico.

- ***Verificación:*** Proceso de evaluar un sistema o componente para determinar si los productos de una fase de desarrollo dada satisface las condiciones impuestas al principio de esa fase.

Referencias Bibliográficas

- Aaen, I., 2003, Software Process Improvement: Blueprints versus Recipes, *IEEE* Software, vol. 20, no. 5, pp. 86-93, Sep./Oct. 2003, doi:10.1109/MS.2003.1231159

- Abran, A., Moore, J. W., Bourque, P., y Dupuis, R. Eds, 2004 Guide to the Software Engineering Body of Knowledge - SWEBOK. *IEEE* Press.

- Agile Spain, 2005. Manifiesto Ágil. En línea. http://www.agile-spain.com/manifiesto_agil

- Ambler, S., 2002, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, John Wiley & Sons, New York.

- Ambler, S., 2003. Agile Database Techniques: Effective Strategies for the Agile Software Developer. Wiley Publishing, USA.

- Ambler, S., 2005, The Agile Unified Process (AUP). En línea. http://www.ambysoft.com/unifiedprocess/agileUP.html

- Ambler, S, 2005a, Disciplined Agile Software Development: Definition. En línea. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm

- Ambler, S. 2007. The Discipline of Agile. Dr. Dobbs, ed. Septiembre 2007, en línea, http://www.ddj.com/architect/201804241

- Anderson, D. J., 2005, "Stretching Agile to fit CMMI Level 3 the story of creating MSF for CMMI® Process Improvement at Microsoft Corporation" presentado en Agile 2005 Conference, *IEEE* Computer Society, Denver.

- Arni-Bloch, N., 2005, Towards a CAME for Situational Methods Engineering. En Pre-proceedings of the First International Conference on Interoperability of Enterprise Software and Applications, INTEROP-ESA'05, Ginebra.

- Aydin, M., Harmsen, F., van Slooten, K., Stegwee, R., 2004, An Agile Information Systems Development Method in Use, En Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 12, No 2, Ankara.

- Ayewah, N., Hovemeyer, D., Morgenthaler, J., Penix, J., Pugh, W., 2008. Using Static Analysis to Find Bugs. *IEEE* Software, vol. 25, no. 5, pp. 22-29.

- Barrios, J., 2001. Une Méthode pour la Définition de l'Impact Organisationnel du Changement. PhD thesis. University of Paris I.

- Balbo, S., Khan, A., 2004. A Tale of two methodologies: Heavyweight versus Agile. Proceedings of the 10th Australian World Wide Web Conference AUSWEB'04. En línea. http://ausweb.scu.edu.au/aw04/papers/edited/balbo/

- Bate, R, Shrum, S., 1998. CMM Integration (CMMI) *Framework*. En línea http://www.sei.cmu.edu/library/abstracts/news-at-sei/featuresept98pdf.cfm

- Beck, K., 2000, Extreme Programming Explained: Embrace Change, Addison-Wesley.

- Boehm, B., Turner, R., 2003. Balancing Agility and Discipline: A guide for the perplexed. Addison-Wesley.

- Brinkkemper, S., 1996. Method engineering: engineering of information systems development methods and tools. Information and Software Technology #38, p275-280.

- CMMI *Product Team*, 2006, CMMI® for Development, Version 1.2. CMU/SEI-2006-TR-008. Carnegie Mellon / Software Engineering Institute, Pittsburgh.

- Davis, C., Glover, M., Manzo, J., Opperthauser, D., 2005. An Agile Approach to Achieving CMMI. AgileTek. En línea. http://www.agiletek.com/images/AgileTek/pdf/an_agile_approach_to_achieving_cmmi.pdf

- Dutton, J., McCabe, R., 2005, Agile/ Lean Development and CMMI®. presentado en SEPG 2006, SEI, Tennessee.

- Fowler, M., 2005. The New Methodology. En línea. http://martinfowler.com/articles/newMethodology.html

- Fowler, M., Highsmith, J., 2001, The Agile Manifesto. Dr. Dobbs, ed. Agosto, 2001, en línea http://www.ddj.com/architect/184414755.

- Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 2000. Refactoring: Improving the Design of Existing Code. Addison-Wesley.

- Gibson, D., Goldenson, D., Kost, K., 2006. Performance Results of CMMI®-based Process Improvement. Software Engineering Institute. Technical Report CMU/SEI-2006-TR-004.

- Glazer, H., Dalton, J., Anderson, D., Konrad, M., Shrum, S., 2008. CMMI® or Agile: Why Not Embrace Both! Software Engineering Institute, Technical Note CMU/SEI-2008-TN-003,

- Hamar, V., 2004, Aspectos metodológicos del desarrollo y reutilización de componentes de software. Universidad de de Los Andes, Facultad de Ingeniería, Postgrado en Computación, Mérida.

- Harmsen, A. F., 1997, Situational Method Engineering. Disertación doctoral. Moret, Ernst & Young Management Consultants, Utrecht, Holanda.

- Hefner, R., 2005. Achieving the Promised Benefits of CMMI. Presentado en CMMI Technology Conference & User Group.

- Henderson-Sellers, B., Ralyté, J., 2010. Situational Method Engineering: State-of-the-art Review. Journal of Universal Computer Science, Vol. 16, no. 3, pp. 424-478.

- Hurtado, J., Bastarrica, M., 2006. Implementing CMMI using a Combination of Agile Methods. CLEI Electronic Journal, Volume 9, N° 1, Paper 7.

- *IEEE* Computer Society, 2008, Guide to the Software Engineering Body of Knowledge (SWEBOK). En línea. http://www2.computer.org/portal/web/swebok

- *IEEE* Standards Board, 1990. *IEEE* Std 610-12. Standard Glossary of Software Engineering Terminology.

- *IEEE* Standards Board, 1998, *IEEE* Std 1074-1997. *IEEE* Standard for Developing Software Life Cycle Processes. *IEEE* Computer Society, New York.

- ISO, 2008. ISO/IEC 12207:2008. Systems and software engineering – Software life cycle processes. http://www.iso.org/iso/catalogue_detail?csnumber=43447

- ISO/IEC JTC 1, 2000. ISO/IEC 12207:1995 Software Life Cycle Process. ISO. Stockholm.

- ISO/IEC, 2007. ISO/IEC 42010:2007 Systems and software engineering -- Recommended practice for architectural description of software-intensive systems. ISO.

- Järvi, A., Hakonen, H., Mákilä, 2007. Developer Driven Approach to Situational Method Engineering. in IFIP International Federation for Information Processing. Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S. Henderson Sellers B (Boston Springler) pp. 94-99.

- Jeffries, R, 2001, Extreme Programming. En línea. http://www.xprogramming.com.

- Kruchten, P, 1995. Architectural Blueprints – The "4+1" View Model of Software Architecture. Publicado en *IEEE* Software #12 (6), pp. 42-50.

- Larman, C, 2001, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition), Prentice-Hall.

- Marchesi, M., Succi, G., Wells, D., Williams, L., 2002. Extreme Programming Perspectives. Addison-Wesley, Boston.

- McMahon, P., 2004. Bridging Agile and Traditional Development Methods: A Project Management Perspective. CrossTalk, ed. Mayo 2004. En línea. http://www.stsc.hill.af.mil/crosstalk/2004/05/0405McMahon.html

- Mirbel, I., Ralyté, J., 2006, Situational Method Engineering: combining assembly-based and roadmap-driven approaches. En Requirements Engineering, # 11, pp 58-78.

- Montilva, J. Barrios, J, 2003, A Component-Based Method for developing Web Applications, publicado en Proceedings of the 5th International Conference on Enterprise Information Systems – ICEIS'2003, Angers.

- Montilva, J, Barrios, J, Rivero, M., 2008, Gray Watch, Método de desarrollo de software para aplicaciones empresariales. Proyecto METHODIUS, Mérida, Venezuela.

- Montilva, J. Hamzan, K. Gharawi, M., 2000, The Watch Model for Developing Business Software in Small and Midsize Organizations. *publicado en* Proceedings of the IV World Multiconference on Systemics, Cybernetics and Informatics – SCI'2000, Orlando.

- OMG, 2009. OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.2. En Línea. http://www.omg.org/spec/UML/2.2/Superstructure/PDF/

- Payne, J. E., Alexander R. T., Hutchinson, C. D., 1997. Design-for-Testability for Object-Oriented Software. Object Magazine. 7(5): p 34-43.

- Pikkarainen, M, Mäntyniemi, A., 2006, An Approach for Using CMMI in Agile Software Development Assessments: Experiences from Three Case Studies, *presentado en* SPICE 2006 Conference, ISO, Luxemburg.

- Pohl, K., Dömges, R., Jarke, M., 1994. Decision Oriented Process Modelling. En Proceedings of the 9th International Software Process Workshop, pp 124-128, *IEEE* Computer Society Press.

- Pressman, R., 2002, Ingeniería del Software. Un enfoque práctico. 5ta ed., Mc Graw Hill, Madrid.

- Ralyte, J., 2001, Ingénierie des méthodes à base de composants. Tesis Doctoral. Universidad de Paris 1 – Sorbonne, Paris.

- Ralyte, J., Deneckère, R., Rolland, C., 2003, Towards a Generic Model for Situational Method Engineering. En Proceedings of the conference CAiSE'03. Eder, J., Missikoff, M., Springer-Verlag, Velden-Austria.

- Rivero, D., Montilva, J., Granados, G., Barrios, J., Besembel, I., Sandia, B., 2007, La Industria de Software en Venezuela: Una caracterización de su recurso humano. En Losavio, F et al (EDS). Actas del X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS'07) y del Primer Encuentro Venezolano sobre Tecnologías de Información e Ingeniería de Software (EVETIS'07), pp. 435-443.

- Rolland, C., Prakash, N., 1996. A proposal for context specific method engineering. En Proceedings of the IFIP TC8, WG8.1/8.2 working conference on method engineering, pp 191-208. Atlanta, USA.

- Rolland C., 1997, A primer for method engineering. Proceedings of the conference INFORSID, Toulouse.

- Rolland, C., Prakash, N., Benjamen, A., 1999, A Multi-Model View of Process Modelling. En Requirements Engineering, Springer-Verlag London, #4, pp 169-187.

- Rolland, C., 2005. L'Ingénierie des Méthodes - Une Visite Guidée. e-TI La Revue Électronique des Technologies d'Information, en línea, http://www.revue-eti.net/document.php?id=726

- Schwaber, K, 2004. Agile Project Management with Scrum. Microsoft Press, Redmond, USA.

- Serour, M.K., Henderson-Sellers, B., 2004, Introducing Agility: A Case Study of Situational Method Engineering Using the OPEN Process *Framework*. En Proceedings of the 28th Annual International Computer Software and Application Conference (COMPSAC'04). *IEEE* Computer Society.

- Sheard, S., 2003. Life Cycle of a Silver Bullet. CrossTalk, ed. Julio, 2003. En línea. http://www.stsc.hill.af.mil/crosstalk/2003/07/sheard.html

- Software Engineering Institute, 2009. CMMI Appraisals FAQ. En línea. http://www.sei.cmu.edu/cmmi/start/faq/appraisals-faq.cfm

- Sommerville, I, 2005. Ingeniería del Software, 7ma ed. Pearson Addison Wesley, Madrid.

- Tolvanen, J., 1998, Incremental Method Engineering with Modelling Tools: Theoretical Principles and Empirical Evidence, Disertación Doctoral, University of Jyväskylä.

- van Slooten, K. and Hodes, B. 1996. Characterizing IS development projects. In *Proceedings of the IFIP Tc8, Wg8.1/8.2 Working Conference on Method Engineering on Method Engineering: Principles of Method Construction and Tool Support: Principles of Method Construction and Tool Support* (Atlanta, Georgia, United States). S. Brinkkemper, K. Lyytinen, and R. J. Welke, Eds. Chapman & Hall Ltd., London, UK, 29-44.

- Weber, A., 2002. Going to Extremes. Dr. Dobbs, ed. Enero, 2002. En línea: http://www.drdobbs.com/windows/184414801

- Wells, D, 2006, Extreme Programming: A gentle introduction. http://www.extremeprogramming.org