Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla
Avda Reina Mercedes, s/n. 41012 SEVILLA
Fax : 95 455 71 39. Tlf: 95 455 71 39. E-mail: lsi@lsi.us.es

Estudio comparativo de propuestas para la generación de casos de prueba a partir de requisitos funcionales.

Javier Jesús Gutiérrez, María José Escalona, Manuel Mejías, Jesús Torres

{javierj, escalona, risoto, torres}@lsi.us.es

Universidad de Sevilla

Lenguajes y Sistemas Informáticos

España

Sevilla, Febrero de 2005

HOJA DE CONTROL DE MODIFICACIONES

Número	Fecha	Descripción	Autores
0	11 – 2004	Desarrollo Inicial	Javier Jesús Gutiérrez y María José Escalona.
1	02 – 2005	Versión final	Los autores del trabajo.

Índice.

1. Introducción.

Hoy en día, debido al aumento del tamaño y la complejidad del software, el proceso de prueba se ha convertido en una tarea vital en el de desarrollo de cualquier sistema informático. Son muchas las clasificaciones de los tipos de prueba que se pueden realizar. Una clasificación posible es la propuesta por Métrica 3 [15] y que se muestra en la tabla 1.

Tipo de pruebas	Fase de realización	Descripción
Pruebas Unitarias.	Durante la construcción del sistema	Prueban el diseño y el comportamiento de cada uno de los componentes del sistema una vez construidos.
Pruebas de Integración.	Durante la construcción del sistema	Comprueban la correcta unión de los componentes del sistema entre sí a través de sus interfaces, y si cumplen con la funcionalidad establecida
Pruebas de Sistema.	Después de la construcción del sistema	Prueban a fondo el sistema, comprobando su funcionalidad e integridad globalmente, en un entorno lo más parecido posible al entorno final de producción.
Pruebas de Implantación.	Durante la implantación en el entrono de producción.	Comprueba el correcto funcionamiento del sistema dentro del entorno real de producción.
Pruebas de Aceptación.	Después de la implantación en el entorno de producción.	Verifican que el sistema cumple con todos los requisitos indicados y permite que los usuarios del sistema den el visto bueno definitivo.
Pruebas de Regresión.	Después de realizar modificaciones al sistema.	El objetivo es comprobar que los cambios sobre un componente del sistema, no generan errores adicionales en otros componentes no modificados.

Tabla 1 . Descripción de las pruebas del software.

En la ilustración 1 se muestra sobre que elementos se realiza el conjunto de pruebas de la tabla 1 y quienes son los participantes.

Las pruebas unitarias y de integración no pueden comenzar hasta que no se dispone de componentes del sistema ya construidos, al igual que las pruebas de implantación y aceptación que tampoco pueden comenzar hasta que se disponga del sistema completo y se instale en su entorno de producción. Sin embargo, la planificación de las pruebas de sistema puede comenzar antes de que el sistema esté terminado. Como el objetivo de las pruebas de sistema es comprobar que todo lo que se está desarrollando cumple con los requisitos del sistema, la planificación de estas pruebas y el diseño de los casos de prueba pueden comenzar tan pronto como empiecen a estar disponibles las especificaciones funcionales del sistema.

La planificación y diseño de las pruebas de sistema en las primeras fases de desarrollo, cuando aún se realiza la elicitación de requisitos, permite encontrar errores, omisiones, inconsistencias y sobreespecificaciones en los requisitos funcionales cuando

aún es fácil y económico corregirlas, ya que el coste de eliminar defectos aumenta a medida que aumenta el tiempo qué transcurre entre la aparición del defecto y su detección [13].

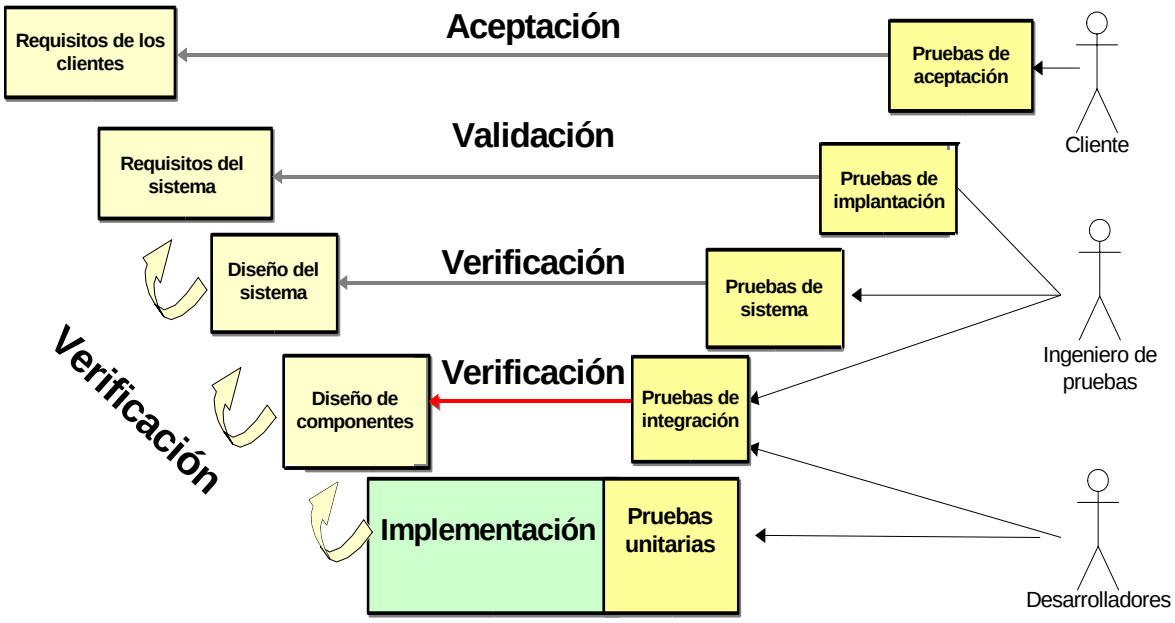

Ilustración 1. Descripción de las pruebas del software.

Para que el proceso de prueba del sistema sea eficaz debe estar integrado dentro del propio proceso de desarrollo y, como cualquier otra fase de dicho proceso, debe realizarse de manera sistemática, minimizando el factor experiencia o intuición. Esto se puede conseguir a través de metodologías que guíen el proceso de desarrollo de pruebas de sistema a partir de requisitos funcionales.

1.1. Objetivos de este trabajo.

El control de calidad de cualquier producto debe comprobar que la calidad del producto final se ajusta a una especificación dada. En el caso concreto del software, el control de calidad debe comprobar que el producto final funcione correctamente de acuerdo con sus especificaciones y en colaboración con otros sistemas software y bases de datos [13].

Un aspecto crucial del control de la calidad son las pruebas del software y, dentro de estas, las pruebas del sistema.

Cuando hablamos de pruebas de software nos referimos a la verificación dinámica del comportamiento de un sistema. Esta verificación está basada en la observación de un conjunto seleccionado de ejecuciones controladas o casos de prueba.

Las primeras propuestas aparecidas para la sistematización y automatización del proceso de pruebas de sistema estaban basadas en la utilización de máquinas de estados finitos como herramienta de representación del comportamiento del sistema. Posteriormente, con el auge del conjunto de diagramas propuestos en la notación UML para el modelado de los diferentes aspectos de un sistema, surgieron nuevas propuestas metodológicas y herramientas para el desarrollo del proceso de prueba a partir de escenarios de uso del sistema descritos mediante diagramas, principalmente con la notación UML.

Actualmente, está surgiendo un nuevo grupo de propuestas que intenta integrar estas dos aproximaciones, tanto la aproximación basada en máquinas de estados como la aproximación basada en escenarios de uso, adoptando lo mejor de ambas [14]. Dentro de este grupo de propuestas, tanto SCENT [2] como AGEDIS [5] son dos de sus representantes más destacados.

Las pruebas del sistema son aquellas que se aplican al producto final, las cuales permiten detectar en que puntos el producto no cumple sus especificaciones. El objetivo de estas pruebas es probar a fondo el sistema, comprobando su funcionalidad e integridad globalmente, en un entorno lo más parecido posible al entorno final de producción.

El objetivo de las pruebas del sistema es comprobar que el sistema software que se está desarrollando cumple con la funcionalidad recogida en casos de uso o escenarios.

En este trabajo se describen y analizan cinco propuestas para la obtención de casos de prueba para la realización de pruebas del sistema a partir de los requisitos del sistema. También se analizan los casos prácticos documentados donde se han aplicado estas propuestas y los resultados obtenidos y, por último, se desarrolla un ejemplo práctico con cada una de estas propuestas.

2. Descripción de propuestas.

En este apartado vamos a describir brevemente cinco propuestas que permiten obtener un conjunto de casos de prueba para la realización de pruebas del sistema a partir de los requisitos funcionales del sistema software. En los dos siguientes apartados se estudiaran los casos prácticos a los que se han aplicado y se desarrollará un ejemplo con cada propuestas y en el apartado cinco expondremos las similitudes y diferencias entre ellas y sus principales puntos fuertes y débiles.

Los criterios para la selección de las propuestas analizadas han sido, en primer lugar, seleccionar las propuestas más actuales. En este aspecto ninguna de las propuestas analizadas en este artículo es anterior al año 2.002, e incluso una de ellas, AGEDIS [5], ha finalizado a principios del año 2.004 por lo que es la propuesta más actual existente en el momento de preparar este artículo.

En segundo lugar hemos buscado propuestas basadas en la corriente actual de aunar las aproximaciones de máquinas de estados y escenarios de uso, por lo que hemos seleccionado dos de las propuestas más representativas en este campo como son SCENT [2] y AGEDIS.

En tercer lugar hemos seleccionado, como alternativa a las dos propuestas anteriores, dos propuestas sencilla y rápidas de poner en marcha, una llamada *Generación de casos de prueba a partir de casos de uso* [1] y otra llamada *UML-Based Statistical Test Case Generation* [8].

Las cuatro propuestas seleccionadas en los párrafos anteriores obtienen un conjunto de pruebas funcionales. Sin embargo, existen más tipos de pruebas posibles, como pruebas de seguridad, pruebas de carga y rendimiento, etc. Por este motivo hemos seleccionado una propuesta que nos permite derivar casos de prueba para realizar pruebas de fiabilidad llamada *UML-Based Statistical Test Case Generation* [4].

2.1. Method Employing Scenarios to Systematically Derive Test Cases for System Test (SCENT).

SCENT [2], [3] es una metodología que se puede dividir en dos grandes bloques. En el primero, SCENT describe un proceso para definir escenarios de uso, refinarlos y organizarlos, a partir de los requisitos funcionales del sistema software. En el segundo, a partir de estos escenarios, describe como obtener sistemáticamente casos de prueba de sistema.

2.1.1. Bloque 1. Obtención de escenarios de uso.

SCENT define escenario de uso como un conjunto ordenado de interacciones entre un sistema y uno o varios actores. Un escenario puede abarcar una secuencia concreta de interacciones o un conjunto de posibles interacciones o caminos de ejecución. Los pasos

para obtener un conjunto de escenarios adecuados para la posterior derivación de casos de prueba se muestran en la tabla 2.

Paso	Descripción	Resultado
1	Encontrar todos los actores que interactúan con el sistema.	Lista de actores
2	Encontrar todos los eventos externos relevantes para el sistema (disparadores / triggers).	Lista de eventos.
3	Determinar las entradas y salidas del sistema y sus resultados.	Entradas y salidas del sistema.
4	Determinar los límites del sistema	Diagrama de contexto.
5	Crear descripciones de escenarios sin detallar, solo con el nombre y una breve descripción.	Lista de escenarios.
6	Asignar prioridades a los escenarios de acuerdo a su importancia y verificar que los escenarios cubren toda la funcionalidad del sistema.	Lista de escenarios ordenador por prioridad y enlaces escenarios – actores.
7	Para cada escenario detallar paso a paso su secuencia de eventos, organizándolos según la plantilla que SCENT proporciona.	Flujo de acciones en los escenarios.
8	Crear un diagrama de dependencias.	Diagrama de dependencias.
9	Revisión y corrección de los escenarios y diagramas por loa usuarios.	Anotaciones de los escenarios.
10	Extender los escenarios refinando la descripción del flujo de acción principal y descomponer las tareas en pasos simples.	Descripción del flujo principal de acción.
11	Modelar flujos alternativos de ejecución, indicando las excepciones que provocan y como reaccionar a estas excepciones.	Flujos de ejecución alternativos, excepciones y su manejo.

12	Factorizar y obtener escenarios abstractos con los elementos comunes de grupos de escenarios.	Escenarios abstractos.
13	Incluir requisitos no funcionales, de calidad y de rendimiento en los escenarios.	Escenarios con requisitos no funcionales.
14	Revisar los gráficos de dependencias.	Diagramas revisados.
15	Revisión y validación de los escenarios por parte de los usuarios (revisión formal).	Escenarios validados.
16	Estructurar los escenarios de acuerdo a la plantilla propuesta por SCENT. Crear casos de pruebas preliminares y escribir el plan y especificaciones de pruebas	Documento de especificación de los escenarios.

Tabla 2 . Pasos para obtener escenarios válidos para la derivación de casos de prueba.

El primer paso para la obtención de escenarios de uso, como indica la tabla 2, es identificar todos los actores que interactúan con el sistema y sus roles y determinar todos los eventos relevantes para el sistema y los resultados esperados. Con estos elementos se construyen escenarios de uso genéricos y se les asignan prioridades en función de su importancia. A continuación se describen detalladamente estos escenarios y se elabora un diagrama de dependencias entre escenarios, en el que quedan reflejados cuáles deben realizarse antes de un escenario concreto, o cuales se pueden realizar independientemente.

En la ilustración 2 se muestra un ejemplo de diagrama de dependencias entre cinco escenarios de uso basados en un sistema de préstamo de libros. En este ejemplo se aprecia como uno de los escenarios, “Borradote usuario” es independiente de los demás y puede realizarse en cualquier momento. Los cuatro restantes están relacionados entre sí. El escenario “Préstamo de libros” solo puede realizarse siempre que se hayan realizado primero los escenarios de “Registro de usuario” y “Registro de libro”, en cualquier orden. El escenario “Préstamo de libros”, puede realizarse varias veces mientras el número de libros en préstamo sea inferior a 4. Por último el escenario de “Devolución de libros” solo puede realizarse después de realizarse el escenario de “Préstamo de libros”.

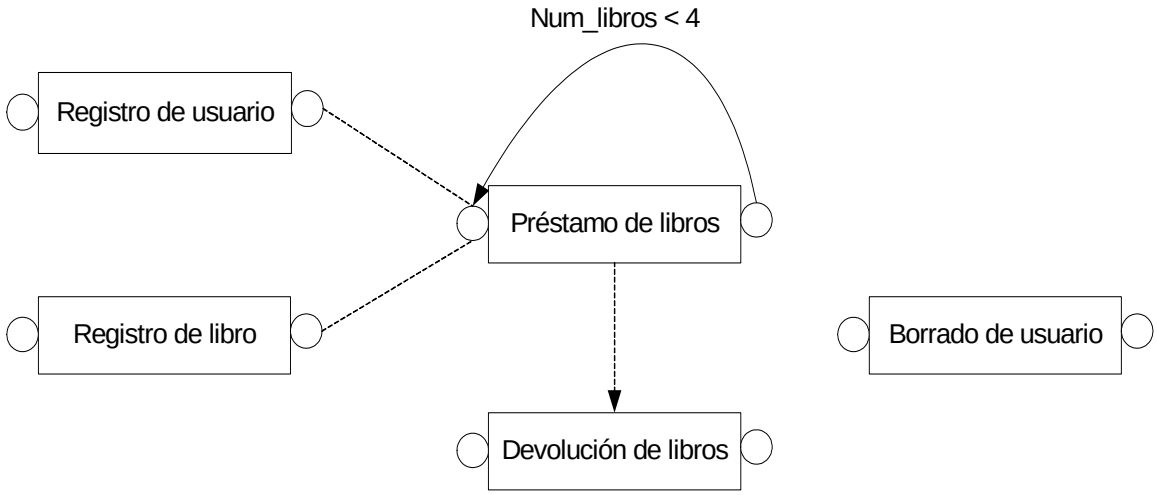

Ilustración 2. Ejemplo de diagrama de dependencias entre escenarios para una sencillo sistema de préstamo.

Después, se modelan flujos alternativos de ejecución, indicando cómo debe reaccionar el sistema ante estos flujos, y se añaden a los escenarios. A continuación se incluye en los escenarios de uso requisitos no funcionales, como descripciones de interfaces de usuario o notas sobre rendimiento. Por último se validan los escenarios de uso obtenidos para asegurar que los escenarios y todos los productos derivados reflejen adecuada y completamente las necesidades de los usuarios del sistema.

Una vez validados, los escenarios de uso son traducidos a diagramas de estados. Estos diagramas de estados se completan con toda la información necesaria para poder generar casos de prueba, como precondiciones, entradas y salidas de datos y requerimientos no funcionales.

Al final de este bloque se obtiene un conjunto de escenarios de uso y diagramas de estados con la información necesaria para la generación de casos de prueba.

2.1.2. Bloque 2: Generación automática de casos de prueba a partir de escenarios.

La generación de casos de prueba a partir de los escenarios de uso y diagramas de estados obtenidos en el bloque anterior se realiza mediante un proceso de tres pasos que se resumen en la tabla 3.

En el primer paso se define el caso de prueba, indicando que es lo que se va a probar. En el segundo paso se generan los casos de prueba a partir de los distintos caminos que recorren a cada diagrama de estados. En el tercer y último paso se refinan los casos de prueba obtenidos y se mejoran con más casos de prueba. Estos casos de prueba adicionales se desarrollan mediante métodos de prueba clásicos, como pruebas de carga y esfuerzo, pruebas de la interfaz de usuario o de las bases de datos, etc.

El único paso obligatorio es el paso 1. Tanto el paso 2 como el paso 3 son opcionales y pueden omitirse si se considera que el conjunto de pruebas obtenido mediante el paso 1 es suficientemente completo.

Paso	Descripción	Resultado
1	Derivar casos de prueba a partir de diagramas de estados.	Conjunto de casos de prueba a partir de cada uno de los diagramas de estados.
2	Derivar casos de prueba para probar dependencias entre escenarios y pruebas adicionales.	Conjunto de casos de prueba a partir de un conjunto de diagramas de estados.
3	Integración de diagramas de estados y derivación de casos de prueba a partir del diagrama de estados resultante.	Conjunto refinado de casos de prueba.

Tabla 3. Pasos para obtener casos de prueba a partir de escenarios válidos.

2.2. Generación de casos de prueba a partir de casos de uso.

Esta propuesta [1] desarrolla un método basado en tres pasos para obtener un conjunto de casos de prueba del sistema a partir de casos de uso en UML.

Paso	Descripción	Resultado
1	Generar escenarios de uso.	Todos los posibles caminos de ejecución de cada caso de uso. Cada camino es un escenario de uso.
2	Identificar casos de prueba.	Conjunto de casos de prueba a partir de los escenarios anteriores.
3	Identificar los valores a probar.	Valores de prueba asociados a cada caso de prueba anterior.

Tabla 4. Descripción de los pasos para la obtención de un conjunto de pruebas de sistema a partir de casos de uso.

Para generar los casos de prueba primero se generan todos los posibles escenarios, o caminos de ejecución, de cada casos de uso. En la ilustración 3 se muestra un ejemplo genérico donde se aprecia un camino de ejecución principal, flecha de color oscuro, y cuatro caminos de ejecución alternativos, uno que provoca la finalización de la ejecución y tres que hacen retroceder la ejecución a un paso anterior.

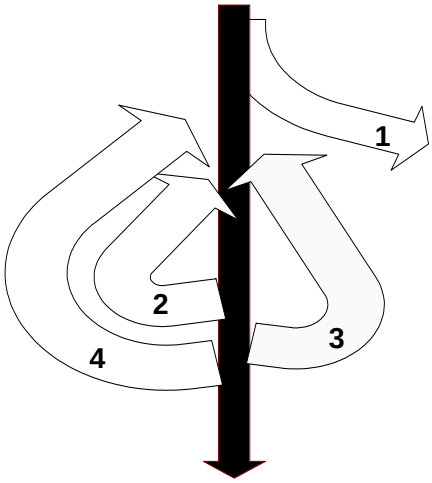

Ilustración 3. Ejemplo de un camino de ejecución principal y cuatro caminos alternativos.

Después, se identifican los casos de prueba a partir de esos escenarios y, por último, se identifican los valores a probar de cada caso de prueba.

Tomando como punto de partida los casos de uso y su descripción no formal obtendremos al final una lista de casos de prueba, con los valores que deben probar y los resultados esperados para cada caso.

2.3. UML-Based Statistical Test Case Generation.

Esta propuesta, a diferencia de las demás, está centrada en pruebas de uso estadístico o pruebas de fiabilidad, las cuales verifican que el sistema cumple un nivel de fiabilidad determinado. La idea principal es que diferentes partes de un programa no necesitan ser probadas con la misma minuciosidad porque el 90% del tiempo se consume en ejecutar sólo un 10% del código [4]. Las pruebas de uso estadístico identifican el código correspondiente a ese 10% y comprueban su fiabilidad.

Paso	Descripción	Resultado
1	Refinado de los casos de uso	Casos de uso refinados con precondiciones, postcondicones, etc.
2	Transformación de los casos de uso a diagramas de estado.	Un diagrama de estados por cada caso de uso.
3	Transformación de los diagramas de estados en diagramas de uso.	Diagramas de uso.
4	Transformación de los diagramas de uso en modelos de uso.	Modelos de uso.
5	Transformación de modelo de uso a casos de prueba	Conjunto de casos de prueba aleatorios.

Tabla 5. Descripción de los pasos para lo obtención del conjunto de pruebas.

El proceso de obtención de pruebas de fiabilidad consta de cinco pasos resumidos en la tabla 5.

En primer lugar se refinan los casos de uso ampliándolos con precondiciones y poscondiciones, alternativas al camino de ejecución principal y referencia a otros casos de uso relacionados. Después se traducen a diagramas de estado y se elabora el modelo de uso donde se indica la probabilidad de que ocurra una transición (ilustración 4) y se identifican los caminos de ejecución más frecuentes.

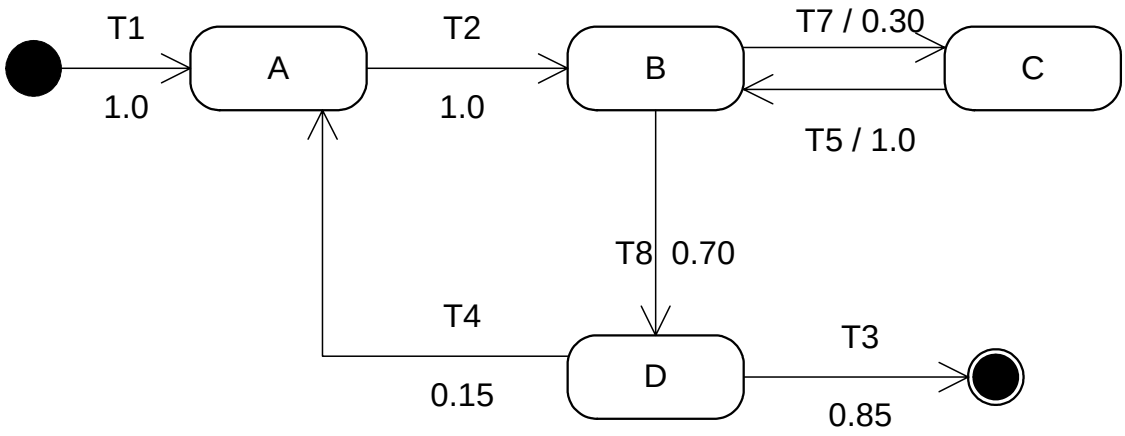

Ilustración 4. Ejemplo de diagrama de estados con la probabilidad de que ocurra cada transición.

Un diagrama de uso utiliza la notación de los diagramas de estado UML, con un único punto de comienzo, un único punto de finalización, y con los estados etiquetados con la acción del usuario que permite cambiar de un estado a otro o épsilon, si la transición no requiere ningún estímulo externo.

Un modelo de uso es un diagrama de uso al que se le ha añadido la probabilidad de que suceda cada transición posible en un estado.

Por último, se extraen los modelos de prueba a partir de los modelos de uso y se generan recorridos aleatorios sobre cada modelo de uso. Cada camino aleatorio será un caso de prueba. Estos casos de prueba, deben elegir qué transición tomarán en función de la probabilidad; por ello, la mayoría de los casos los de prueba recorrerán las transiciones con mayor probabilidad.

El primer caso de prueba que es necesario probar es aquel que recorre el camino mínimo, sin tener en cuenta las transiciones épsilon.

Un caso de prueba aleatorio es un caso de prueba que recorre el modelo de uso eligiendo en cada momento la transición que va a adoptar a partir de las probabilidades de todas las transiciones posibles en el estado en que se encuentre.

2.4. AGEDIS.

2.4.1 Descripción del proceso.

AGEDIS [5], [6] ha sido un proyecto de investigación financiado por la Unión Europea finalizado en el último trimestre del 2.003. Su objetivo ha sido el desarrollo de una metodología del mismo nombre y de un conjunto de herramientas para la generación automática y ejecución de grupos de pruebas para sistemas basados en componentes distribuidos. Aunque la metodología y herramientas pueden aplicarse a cualquier tipo de sistema, está más orientada a sistemas de control distribuidos, como protocolos de comunicaciones con un sistema servidor y varios sistemas clientes, que a sistemas de transformación de la información, como compiladores.

AGEDIS se centra en dos productos: el primero está compuesto de un modelo del sistema escrito en el lenguaje de modelado IF y un conjunto de diagramas UML de clases y estados que van a permitir la generación automática del conjunto de pruebas. El segundo está compuesto de un conjunto de objetos de casos de prueba ejecutables tanto en el modelo del sistema como en la implantación, lo que permite comparar los resultados esperados, los del modelo, con los obtenidos, los del sistema.

La metodología propone un proceso iterativo de seis pasos que se resumen en la tabla 6.

Paso	Descripción	Resultado
1	Elaboración del modelo del sistema.	Modelo de comportamiento del sistema.
2	Creación de directivas de pruebas.	Directivas de generación y ejecución de pruebas.
3	Generación del catálogo de pruebas.	Catálogo de pruebas.
4	Revisión del catálogo de pruebas.	Catálogo de pruebas revisado.
5	Ejecución de las pruebas.	Resultados de las pruebas.
6	Análisis de las pruebas.	Opcionalmente modelo de comportamiento modificado y nuevas directivas de generación y ejecución de prueba

Tabla 6. Pasos para lo obtención de un conjunto de pruebas.

En primer lugar se construye un modelo de comportamiento del sistema a partir de sus especificaciones. Este modelo está compuesto por diagramas UML de clases y un diagrama UML de estados por cada clase que describe el comportamiento de los objetos de dicha clase. A continuación se elaboran los objetivos de las pruebas (pruebas de casos de uso con datos concretos, pruebas de carga del sistema, etc.) y se traducen a un conjunto de directivas de generación y ejecución de pruebas. En el siguiente paso, una herramienta genera automáticamente una serie de pruebas que satisfacen los objetivos de prueba anteriores y se ejecuta automáticamente. En la ilustración 5 se muestra de forma gráfica la ejecución de estos pasos y los resultados obtenidos.

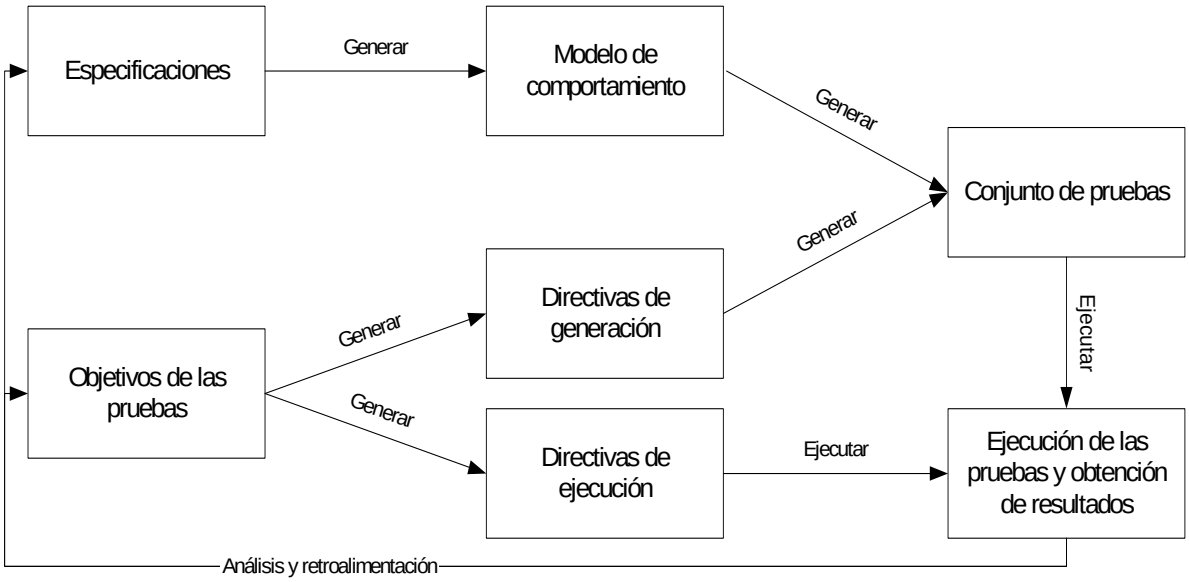

Ilustración 5. Descripción del proceso de obtención y ejecución de casos de prueba.

Por último se analizan los resultados y se repiten los pasos hasta que se alcanzan los objetivos deseados.

2.4.2. Descripción de las herramientas.

El proceso de AGEDIS está soportado por un conjunto de herramientas con una interfaz gráfica común. En la ilustración 6 se muestra de manera gráfica las herramientas que intervienen en cada una de las partes del proceso.

A continuación se describen las herramientas existentes.

- Una interfaz gráfica de usuario común para todas las herramientas de AGEDIS.
- Una herramienta de simulación de modelos que permite depurar el modelo de comportamiento.
- Un generador de pruebas capaz de crear un conjunto de casos de prueba que satisfagan el criterio de cobertura, indicado en los objetivos de la prueba,
- Un marco de ejecución de pruebas independiente de la plataforma y lenguaje del sistema (actualmente soporta C, C++ y Java)
- Dos herramientas para el análisis de los resultados de las pruebas, un analizador de cobertura que informa de valores posibles no probados y caminos de ejecución no recorridos, y un analizador de defectos que agrupa los defectos encontrados en un camino de ejecución.

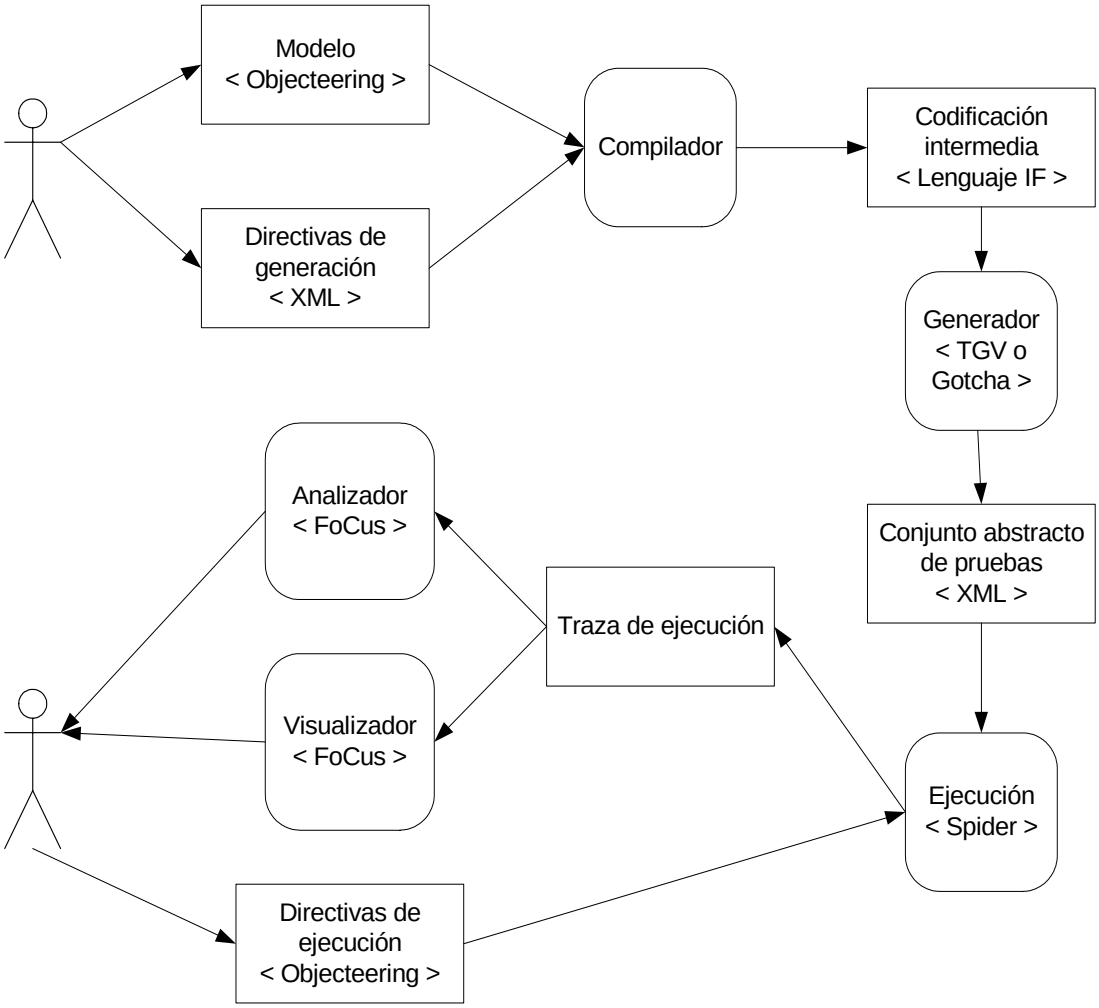

Ilustración 6. Descripción de la arquitectura de herramientas de AGEDIS.

En la tabla 7 se describen brevemente las herramientas de soporte del proceso AGEDIS.

Herramienta.	**Descripción.**
Objecteering	Herramienta para construir modelos de comportamiento del sistema en lenguaje AML (AGEDIS Modelling Language). Se encarga de traducir los modelos en AML a lenguaje IF.
Spider	Motor de ejecución de las pruebas encargado de ejecutar el conjunto de pruebas generado teniendo en cuenta las directivas de ejecución.
FoCus	Herramienta para la visualización y análisis de los resultados de los catálogos de pruebas.
Gotcha y TGV	Motores de generación de pruebas a partir de la descripción del comportamiento del sistema y de las directivas de generación.

Tabla 7. Descripción de las herramientas.

2.5. Use Case Path Análisis.

Esta propuesta [8] describe un proceso de 5 pasos que permite obtener, a partir de un caso de uso, un conjunto de caminos de ejecución adecuado para ser traducidos a pruebas del sistema. Estos pasos se resumen en la tabla 8.

Paso	Descripción	Resultado
1	Elaboración de diagramas de flujo a partir de los casos de uso.	Un diagrama de flujo por cada caso de uso.
2	Identificación de todos los posibles caminos de ejecución.	Listado de caminos de ejecución de cada diagrama de flujo.
3	Análisis y baremación de los caminos.	Listado de caminos baremados.
4	Selección de caminos a probar.	Listado de caminos que se convertirán en casos de prueba.
5	Elaboración de los casos de prueba a partir de los caminos seleccionados.	Un caso de prueba por cada camino.

Tabla 8. Descripción de los pasos para la obtención de casos de prueba.

El primer paso consiste en dibujar un diagrama de flujo para cada caso de uso (Ilustración 7). Cada caso de uso tiene varios caminos de ejecución y cada camino es un caso de prueba potencial.. Este diagrama de flujo, que sigue una notación propia descrita en la propuesta, muestra todos los posibles caminos de ejecución del caso de uso.

El segundo paso consiste en identificar todos los caminos posibles de ejecución. De cada camino se indica su identificador, el nombre del camino formado por los números o nombres de las ramas que cruza y descripción del camino donde se resume, en lenguaje natural, la secuencia completa de iteración del usuario con el sistema.

Después se analizan y puntúan los caminos evaluando sus atributos. Esta propuesta deja abierta la posibilidad de analizar tantos atributos como se considere conveniente y propone dos atributos como ejemplo: la frecuencia con que se ejecuta el camino y el factor crítico del camino.

El atributo frecuencia toma un valor entre 1, un camino muy poco frecuente y, 10, un camino muy frecuente y puntúa la repetición de un camino por parte de un usuario. Los caminos de ejecución más utilizados por los usuarios son los que obtendrán más puntuación en frecuencia.

El atributo Factor crítico también toma un valor comprendido entre 1 y 10, y describe como de crítico puede ser el fallo de este camino. Un fallo en un camino que impida seguir trabajando con el sistema tendrá un valor muy alto en este atributo, mientras que un fallo en un camino que sea recuperable por el sistema con un gasto en recursos mínimo y sin ocasionar ninguna pérdida en la información del sistema tendrá un valor muy bajo.

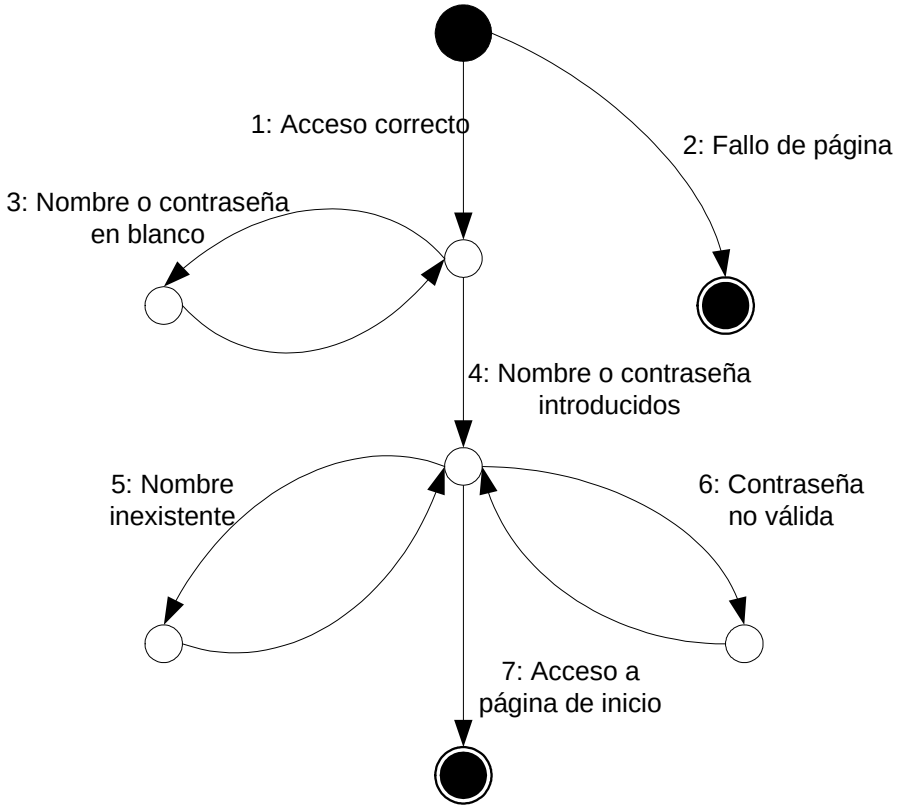

Ilustración 7. Ejemplo de diagrama de flujo de una página Web de login.

La propuesta permite que se introduzcan nuevos atributos si se considera necesario y que se modifique la escala en que a cada uno se le asigna un valor.

Una vez puntuados todos los atributos se obtiene, a partir de ellos, la valoración del camino, o factor del camino. La fórmula incluida en esta propuesta se detalla a continuación, en la tabla 9, aunque también se permite modificarla, por ejemplo asignándole un peso a cada atributo en función de su importancia.

```
Factor del camino = Frecuencia + Factor crítico
```

Tabla 9. Fórmula para la obtención del factor o valoración del camino de ejecución.

Con esta valoración se seleccionan los caminos más significativos obteniendo un conjunto con el mínimo número caminos que garantice que se prueba completamente la funcionalidad crítica y que asegure que todas las ramas de ejecución han sido incluidas en, al menos, un camino. Cada camino seleccionado se convertirá en un caso de prueba.

Por último para camino se obtiene su conjunto de escenarios de prueba. Esta propuesta define escenario de prueba como un camino de ejecución y un conjunto de

valores concretos para ese camino. Generalmente el caso de prueba de un camino de ejecución, tendrá más de un escenario de prueba.

2.6. Resumen de las propuestas estudiadas.

Hemos descrito varias propuestas para obtener pruebas del sistema a partir de los requisitos en las fases tempranas del desarrollo de software, A continuación, en la tabla 10, se ofrece un breve resumen de cada una.

Propuesta.	**Descripción.**
SCENT [2], [3]	Ofrece un método muy completo para manipular y organizar los requisitos funcionales en escenarios de uso y derivar casos de prueba del sistema a partir de ellos.
AGEDIS [5], [6].	Abarca tanto la elaboración de un prototipo del sistema como la generación y ejecución de las pruebas. Cuenta, además, con un conjunto de herramientas para automatizar el proceso.
Generating Test Cases From use Cases [1].	Desarrolla un método basado en tres pasos para obtener un conjunto de casos de prueba del sistema a partir de casos de uso descritos en lenguaje natural.
UML-Based Statistical Test Case Generation [4]	Está orientada a obtener un conjunto de casos de prueba para realizar pruebas de carga en función de las probabilidades de cada caminos de ejecución para cada requisito funcional.
Use Case Path Anmalisys [8].	Analiza todos los caminos de ejecución para un caso de prueba, los puntúa y selecciona el conjunto mínimo que deben traducirse a pruebas del sistema.

Tabla 10. Descripción propuestas para la obtención de casos de prueba a partir de requisitos funcionales.

En el apartado 5 se ofrece una comparativa entre estas propuestas sí como sus puntos fuertes y sus puntos débiles.

3. Descripción de casos de uso reales.

Las únicas propuestas que incluyen referencias a casos de uso reales son: SCENT [9] y AGEDIS [11], por tanto estos van a ser los únicos casos de uso reales estudiados en este apartado.

La propuesta Use Case Path análisis menciona dos proyectos basados en tecnología EJB donde ha sido utilizado, sin embargo no se indica ninguna referencia a documentación donde detalle el uso o donde ofrezca conclusiones. Hemos intentado ponernos en contacto con su autor mediante correo electrónico pero, aunque los correos no han sido devueltos, no hemos obtenido contestación.

Tanto la propuesta Generación de casos de prueba a partir de casos de uso [1] como la propuesta UML-Based Statistical Test Case Generation [4] no mencionan ningún caso de uso real donde haya sido aplicado.

3.1. GLASS, aplicación práctica de SCENT.

GLASS [9] es el nombre de un sistema de monitorización remota de sistemas embebidos de ingeniería eléctrica desarrollado por ABB Corporate Research Center en Baden-Daettwil, Suiza. Este sistema está basado en una arquitectura cliente-servidor de tres capas sobre TCP/IP y, más, basado en clientes y servidor web y el protocolo http. En la ilustración 8 se muestra una visión general de la arquitectura del sistema GLASS.

La propuesta SCENT ha sido aplicada a dos subproyectos del sistema GLASS: uno de ellos es un sistema encargado de la monitorización remota de subestaciones en sistemas de distribución de energía eléctrica, el otro subproyecto es un sistema encargado de la monitorización de transformadores de alto voltaje [9]. En la ilustración 9 se muestra una captura del sistema en funcionamiento.

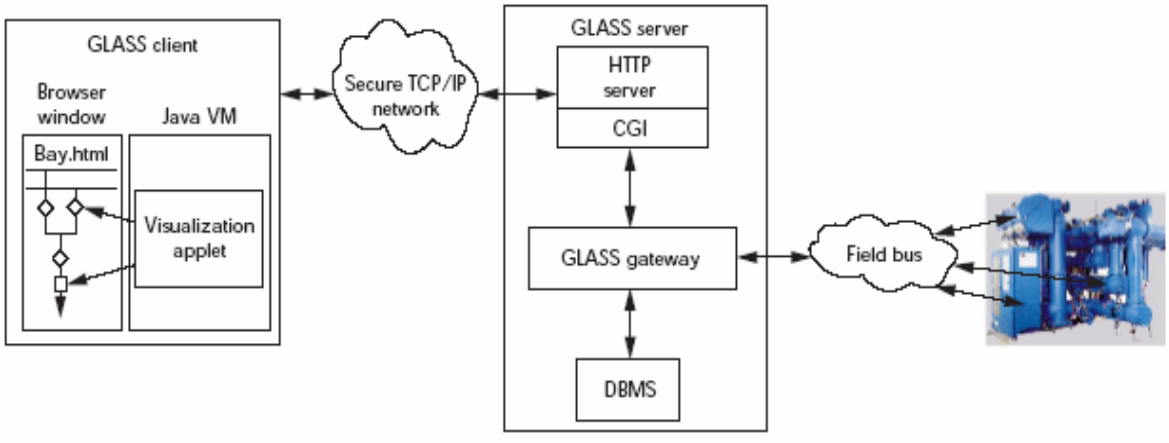

Ilustración 8. Arquitectura del sistema GLASS

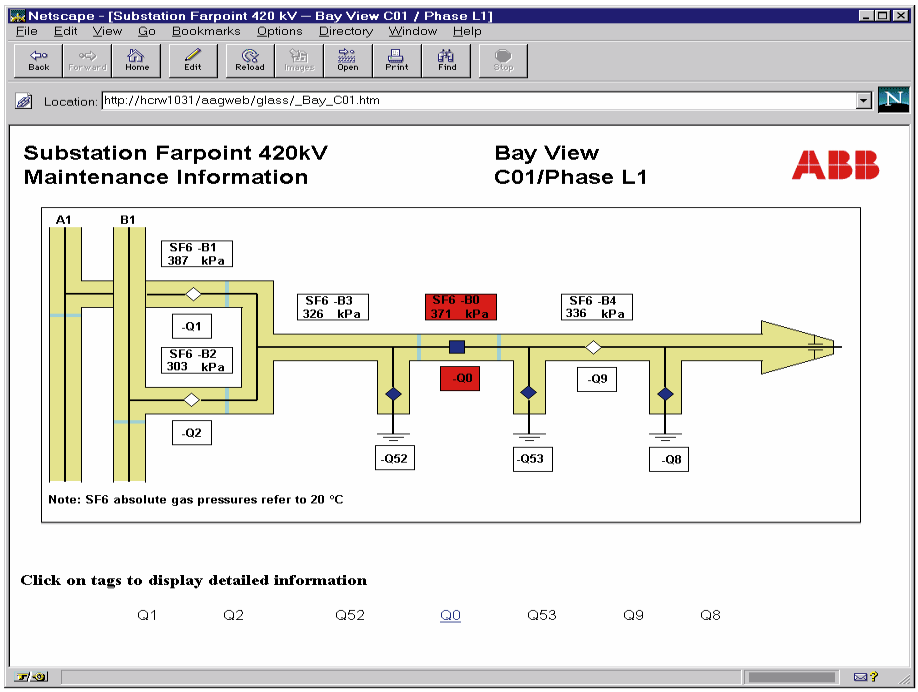

Ilustración 9. Ejemplo de la interfaz Web de GLASS.

Cada subproyecto contaba con su propio grupo de desarrollo. Los desarrolladores de cada uno de los proyectos contaban con experiencia en lenguajes orientados a objetos y con una vaga idea sobre los escenarios y su uso en ingeniería del software.

La formación ofrecida a cada uno de los grupos de desarrollo consistió en un taller de medio día de duración donde se presentó el procedimiento para creación de escenarios descrito en SCENT y se ilustró la formalización de escenarios escritos en lenguaje natural en diagramas de estados mediante un breve ejemplo

Aunque en la documentación, [1] y [9] no se indica ni el número de desarrolladores ni el tiempo invertido en el desarrollo, a juzgar por el número de escenarios, 33 el primer sistema y 14 el segundo, estimamos que los proyectos fueron de tamaño pequeño con un número no superior a 6 u 8 desarrolladores.

3.1.1. Sistema de monitorización remota de subestaciones en sistemas de distribución de energía eléctrica.

En este sistema ya se contaba con una especificación inicial, por lo que solo se desarrollaron partes del método para adaptarse a la implementación existente. En cambio en el segundo, se usó SCENT y su aproximación de escenarios desde el principio.

En este sistema, las pruebas del sistema se restringieron a mostrar las principales características para ser funcional (que hacían al sistema funcional) y garantizaba la obtención de un prototipo operacional para ser usado para demostraciones y formación.

Para este primer sistema fueron necesarios 33 escenarios.

No ha sido necesario un gran periodo de formación. El tiempo empleado en la formación del equipo de desarrollo en el proceso de SCENT fue de un día, con una primera sesión teórica y una segunda sesión práctica. Durante todo el proceso el equipo de desarrollo contó con supervisión de técnicos con un gran conocimiento de SCENT.

3.1.2. Sistema de monitorización de transformadores de alto voltaje.

En este sistema, a diferencia del anterior, se aplicó la propuesta SCENT desde el principio. Sin embargo el tiempo dedicado a la formación de los desarrolladores fue similar al del sistema anterior.

Los requerimientos para el nuevo sistema se recogieron en 14 escenarios.

3.1.3. Conclusiones.

Ambos proyectos concluyeron de forma satisfactoria, siendo SCENT una herramienta fundamental de este éxito.

De los casos prácticos se han extraído las siguientes conclusiones:

- Los clientes apreciaron el uso de escenarios y permitió integrarlos en el proceso creando, refinando y validando escenarios.
- El trabajo de creación y refinado de los escenarios es un proceso iterativo que discurre en paralelo al proceso de formalización.

Algunos de los problemas aparecidos fueron:

- La gestión de escenarios, que al estar escritos en lenguaje natural dificultaba mantenerlos consistentes.
- La granularidad de los escenarios. Distinguir los que es parte de la especificación y la implementación, cuando no es necesario seguir refinando un escenario, etc.
- Dificultad para modelar interacciones internas con los escenarios. Los escenarios están pensados principalmente para modelar interacciones con usuarios.

3.2. AGEDIS.

3.2.1. Descripción de 5 casos de estudio.

Durante los 3 años de desarrollo de AGEDIS se han llevado a cabo 5 casos de estudio reales de la industria. Es cada uno de aplico métodos de prueba basados en modelos y herramientas a problemas reales en la industria. El requisito imprescindible para

la automatización del proceso de obtención y ejecución de los casos de prueba es elaborar un modelo del comportamiento del sistema.

Los dos primeros casos de estudio se realizaron antes de que las herramientas estuvieron disponibles y su principal utilidad fue determinar los requisitos que dichas herramientas debían satisfacer. En concreto, el primer caso de estudio fue llevado a cabo por France Telecom usando la herramienta GOTCHA de IBM para generar conjuntos de pruebas para una implementación del protocolo de comunicaciones Pragmatic General Multicast (PGM). Este caso de estudio se centró en la interacción entre los distintos procesos durante la transmisión de datos.

El segundo caso de estudio se aplicó sobre el proyecto TCP (Transit Computerization Project) para la automatización del intercambio de información realizado por la empresa Intrasoft. Se construyeron dos modelos, uno para su uso con la herramienta TGV y otro con la herramienta GOTCHA para poder comprobar las ventajas y diferencias de cada una y aplicarlas a las herramientas a desarrollar.

El tercer caso de estudio se aplicó sobre el proyecto "WebSphere MQ Telemetry Java Classes", consistente en una interfaz de programación escrita en Java de un protocolo de mensajería. Este fue el primer caso de estudio donde se pudo comprara un modelo desarrollado con las herramientas AGEDIS con modelos desarrollados con la herramienta GOTCHA y RBP, resultando el modelo AGEDIS más apropiado para la comunicación entre desarrolladores y la identificación de las entradas y salidas.

El cuarto caso de estudio se llevó a cabo después de que las herramientas fueran integradas bajo una interfaz gráfica de usuario común a todas ellas. El sistema a probar fue el sistema llamado E-Tendering desarrollado por Intrasoft, el cual proporciona la infraestructura necesaria para servicios de gestión basados en web. Fueron necesarias usar una serie de librerías adicionales para permitir la conexión entre Spider y el sistema que se estaba probando, lo cual demostró que era sencillo realizar esta integración

El quinto y último caso de estudio se aplicó sobre NEMO, un middleware diseñado para distribuir mensajes a una lista de usuarios registrados utilizado como componente en varias aplicaciones de France Telecom. Se desarrolló un modelo muy sencillo, con un único servicio de distribución y dos receptores, sin embargo este modelo demostró que se puede obtener un gran número de casos de prueba válidos y aplicables a modelos de más complejidad a partir de modelos simples y que la vuelta a generar casos de prueba es automática cuando se modifica el modelo.

3.2.2. Conclusiones.

En el desarrollo de pruebas basadas en modelos se comienza a obtener beneficios simplemente por el hecho de moldear. La creación de un modelo sirvió en los casos de estudio para resaltar inconsistencias en las especificaciones y, en varios casos, exponer fallos en fases muy tempranas del proceso de desarrollo.

La naturaleza integrada de las herramientas y sus interfaces fue muy apreciada por los ingenieros, permitiéndoles desarrollar su trabajo de forma rápida y cómoda y aumentando su productividad.

El modelado de un sistema no es una tarea sencilla. Fuerza a los ingenieros a comprender el sistema que están modelando mucho mejor. Gracias a esto las pruebas basadas en modelos pueden detectar inconsistencias en requerimientos y diseño sin necesidad de ejecutar una sola prueba.

Son necesarios más estudios para determinar cuando la construcción de modelos mediante herramientas gráficas, como AML, es mejor que mediante herramientas basadas en texto.

4. Ejemplos de uso.

En este punto vamos a describir con detalle un caso de uso y a derivar un conjunto de pruebas del sistema a partir de él utilizando cada una de las propuestas descritas en el apartado 2.

4.1. Descripción del caso de uso.

Suponemos que, en la fase de requisitos, ya se tiene identificado y validado un requisito funcional llamado "Entrada en el sistema" cuyo camino de ejecución (la secuencia normal y las excepciones posibles) se describen en la tabla 11.

RF-1	**Entrar en el sistema**	
Objetivos	OBJ-01: El sistema debe controlar la entrada en el sistema	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario quiera conectarse al sistema	
Precondición	Ninguna.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario conecta con la página de acceso del sistema.
	2	El sistema solicita autentificación
	3	El sistema recibe la conexión y muestra una página donde pide nombre y contraseña
	4	El usuario rellana el formulario introduciendo su nombre y pulsa el botón entrar
	5	El sistema comprueba el nombre y la contraseña y si ambos son correctos carga la página de inicio
Excepción	**Paso**	**Acción**
	1	[1] Si el servidor no está activo o la página no carga adecuadamente se muestra un mensaje de error y termina el caso de uso.
	4	[2] Si la contraseña o el nombre no se introdujeron, el sistema vuelve a solicitarlos indicando un error.
	4	[3] Si el nombre no se encuentra en la lista de nombres registrados el sistema vuelve a solicitarlo indicando un error
	4	[4] Si la contraseña suministrada no es válida, el sistema lo indica con un error y la pide de nuevo.
Rendimiento	**Paso**	**Cuota de tiempo**
	5	5 segundos.

Tabla 11. Ejemplo de patrón de caso de uso "Entrada en el sistema".

4.2. Generación del conjunto de pruebas aplicando SCENT.

En este ejemplo vamos a centrarnos solamente en el proceso de obtención de casos de prueba. No vamos a ver el proceso de creación y refinamiento de los escenarios, por lo que tomaremos como punto de partida el caso de uso descrito en el apartado 4.1, el cual es lo suficientemente completo para poder aplicar directamente la generación de casos de prueba. A continuación, aplicaremos solo el segundo bloque (apartado 2.1.2.) de la propuesta metodológica y solo el primer punto de la misma, el único punto que es obligatorio.

Después del proceso de generación y refinado de los escenarios habremos obtenido, entre otros productos, un diagrama de estado que represente el escenario. El diagrama de estados para el escenario del apartado 4.1. se muestra a continuación, en la ilustración 10.

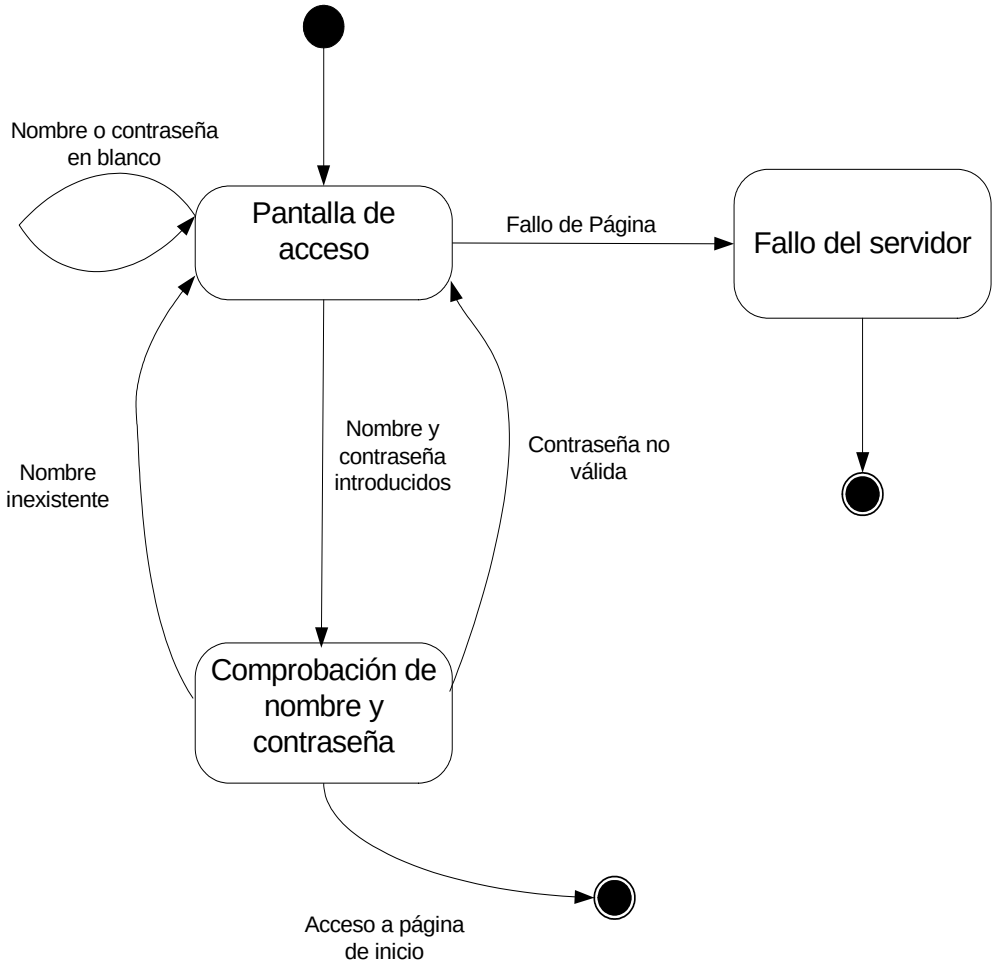

Ilustración 10 . Diagrama de estados correspondiente al caso de uso del apartado 4.1.

Este diagrama de estados es necesario extenderlo con precondiciones, postcondiciones, datos de entrada, los datos de salida esperados y requisitos no funcionales, tales como requisitos de ejecución.

Las precondiciones y postcondiciones ya aparecen en la plantilla de escenario que hemos utilizado, en la tabla 11.

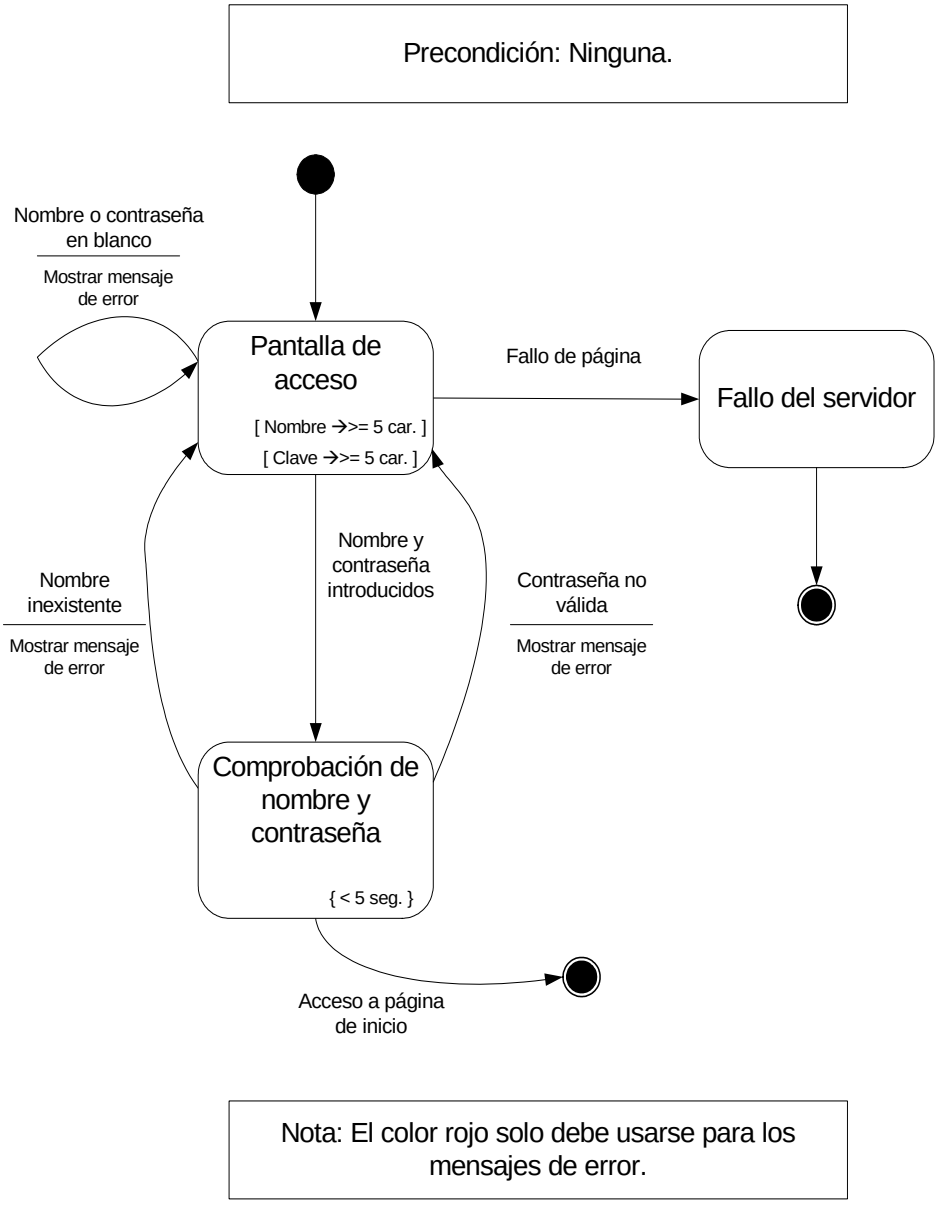

Ilustración 11 . diagra de estados con información extendida.

Como rango de los datos de entrada, establecemos que tanto el nombre de usuario como su clave han de tener una extensión mínima de cinco caracteres. Los rangos se indican dentro de los estados mediante corchetes o llaves o paréntesis, si alguno de estos

símbolos ya se utiliza para algún otro propósito. En este ejemplo concreto, los representaremos entre corchetes.

Los requisitos no funcionales se expresan de la misma manera que los rangos de los datos de entrada. En este ejemplo indicaremos los requisitos no funcionales encerrándolos entre llaves para distinguirlos de los rangos de datos. En el diagrama de estados añadiremos como requisito no funcional la nota sobre rendimiento que recoge que la validación de un nombre y una clave debe realizarse en un tiempo inferior a cinco segundos. También vamos a añadir una anotación para reservar el color rojo solo para mensajes de error.

A continuación se muestra el diagrama de estados con toda la información extendida en la ilustración 11.

Preparación de la prueba:		Existe un usuario válido con nombre de usuario “usuario” y clave “clave”	
Id	**Estado**	**Entrada/Acción del usuario / Condición**	**Salida esperada**
1	Pantalla de acceso	El actor introduce un nombre válido (“usuario”) y su contraseña correspondiente (“clave”).	Página de inicio del usuario.
2	Pantalla de acceso	Se produce un fallo de página.	Mensaje de fallo de página.
3	Pantalla de acceso	El actor introduce un nombre o contraseña en blanco.	Se vuelve a solicitar el nombre de la contraseña.
4	Pantalla de acceso	El actor introduce un nombre menor de 5 caracteres (“usua”).	Se vuelve a solicitar el nombre y la contraseña.
5	Pantalla de acceso	El actor introduce una clavemenor de 5 caracteres (“usua”).	Se vuelve a solicitar el nombre y la contraseña.
6	Comprobación de nombre y contraseña	El sistema valida correctamente el nombre de usuario y la contraseña.	Página de inicio del usuario
7	Comprobación de nombre y contraseña	El sistema valida incorrectamente el nombre de usuario y la contraseña	Mensaje de usuario incorrecto. Se vuelve a solicitar el nombre y la contraseña.
8	Fallo del servidor	Fallo del servidor	Mensaje de fallo de página.

Tabla 12. Casos de prueba del caso de uso.

En SCENT los casos de prueba para pruebas de sistema son derivados a partir de los posibles caminos que recorren del diagrama de estados. En primer lugar se localiza el camino principal del diagrama, el cual será el primer caso de prueba y el más importante (en nuestro ejemplo la fila con Id 1 en la tabla 12). A continuación se localizan los caminos que representan flujos alternativos de ejecución. Mediante este proceso se recorren todos los nodos y todas las transiciones del diagrama con, al menos, un caso de prueba.

Las precondiciones definen los elementos que deben hacerse antes de poder ejecutar el test derivado. La configuración, o *setup*, de la prueba queda determinado por las precondiciones. En nuestro ejemplo concreto, al no existir ninguna precondición, no es necesario realizar ninguna tarea preparatoria antes de la ejecución de las pruebas.

El primer caso de prueba sigue el camino de ejecución normal, sin que suceda ningún fallo: un visitante accede a la página de acceso, introduce su nombre de usuario y clave y accede a su página de inicio. El siguiente caso de prueba se obtiene al considerar la posibilidad de un error del servidor.

En cada camino también se indica con que datos concretos se ha de probar.

La lista de todos los casos de prueba, a partir de los caminos de ejecución posibles, que obtenemos a partir del diagrama de estados anterior se muestra a continuación en la tabla 12.

Estos casos de prueba pueden refinarse incluyendo más información recogida en los requisitos.

Técnicas de prueba de dominio y pruebas de flujos de datos pueden aplicarse para derivar casos de prueba adicionales.

4.3. Generación del conjunto de pruebas aplicando Generating Test Cases from Use Cases.

La parte más importante de un caso de uso para la generación automática de un caso de prueba es el camino de ejecución. Este camino se divide en dos en el patrón: el camino principal o secuencia normal y los caminos alternativos o excepciones. El camino principal son los pasos que da el sistema si no surge ningún imprevisto ni error, mientras que los caminos alternativos son las variaciones que pueden surgir en distintos puntos del camino principal a causa de errores, rectificaciones, etc. A cada uno de estos caminos lo llamaremos escenario de caso de uso. Todos los escenarios de caso de uso posibles serán utilizados como base para crear las pruebas.

Un caso de prueba será un conjunto de entradas con datos de prueba, unas condiciones de ejecución, y unos resultados esperados. Para generar los casos de prueba aplicamos los tres puntos de la propuesta:

En el primer punto identificamos todas las combinaciones posibles de caminos de ejecución del caso de uso, es decir todas las combinaciones posibles entre el camino principal y los caminos alternativos y le asignamos un nombre. Cada combinación será un escenario de uso. Todos los posibles caminos se recogen a continuación en la tabla 13.

Escenarios de caso de uso.	Comienzo	Excepciones	
1. Acceso correcto	Secuencia normal		
2. Fallo de página (servidor no disponible o no carga adecuadamente).	Secuencia normal	Excepción 1	
3. Nombre o contraseña en blanco.	Secuencia normal	Excepción 2	
4. Nombre no existente.	Secuencia normal	Excepción 3	
5. Contraseña no válida(no corresponde con nombre de usuario).	Secuencia normal	Excepción 4	
6. Nombre o contraseña en blanco y después escribir nombre no existente.	Secuencia normal	Excepción 2	Excepción 3
7. Nombre o contraseña en blanco y después escribir clave incorrecta.	Secuencia normal	Excepción 2	Excepción 4
8. Análogo al escenario 6	Secuencia normal	Excepción 3	Excepción 2
9. Nombre no existente y después contraseña incorrecta.	Secuencia normal	Excepción 3	Excepción 4
10. Análogo al escenario 7.	Secuencia normal	Excepción 4	Excepción 2
11. Análogo al escenario 9.	Secuencia normal	Excepción 4	Excepción 3

Tabla 13. Todos los posibles escenarios de uso para el caso de uso en estudio.

Los escenarios de caso de uso del 6 al 11, son redundantes, por los que no los usaremos a la hora de obtener los casos de prueba.

En el segundo punto, estudiamos la descripción del caso de uso de partida y extraemos las condiciones o valores requeridos para la ejecución de los distintos escenarios. Esta información se muestra a continuación en la tabla 14.

ID	Escenarios	Fallo de página	Nombre	Contraseña	Resultado esperado
1	Acceso correcto	No	V	V	Carga de la página de inicio
2	Fallo de página	Sí	N/A	N/A	Mensaje de error.
3	Nombre o contraseña en blanco.	No	Vacío	Vacío	Mensaje de error. El sistema solicita autentificación de nuevo.
4	Nombre no existente.	No	I	N/A	Mensaje de error. El sistema solicita autentificación de nuevo.
5	Contraseña no válida.	No	V	I	Mensaje de error. El sistema solicita autentificación de nuevo.

Tabla 14 Matriz de casos de prueba.

I – Valor inválido.
V – Valor válido.
Vacío – No se indica ningún valor.
N/A – El valor que tenga es irrelevante.

En el tercer punto, ya con todos los casos de prueba identificados, se revisan para asegurar su exactitud y localizar casos redundantes o la ausencia de algún caso. Por último, para cada escenario de caso de uso, se identifican sus valores de prueba. Los casos de prueba con sus valores de pruebas definitivos se recogen en la tabla 15.

ID	Escenarios	Fallo de página	Nombre	Contraseña	Resultado esperado
1	Acceso correcto	HTTP Ok	John	Dough	Carga de la página de inicio
2	Fallo de página	Error 505	N/A	N/A	Mensaje de error.
3	Nombre o contraseña en blanco	HTTP Ok	John “”	“” Dough	Mensaje de error. El sistema solicita autentificación de nuevo.
4	Nombre no existente.	HTTP Ok	Jane	N/A	Mensaje de error. El sistema solicita autentificación de nuevo.
5	Contraseña no válida.	HTTP Ok	John	Anyone	Mensaje de error. El sistema solicita autentificación de nuevo.

Tabla 15 Matriz de casos de prueba con sus valores de prueba.

Con el conjunto escenarios de casos de uso y valores de prueba obtenidos se podrán realizar fácilmente las pruebas del sistema que garanticen que toda la funcionalidad recogida en este caso de uso ha sido correctamente implementada. Repitiendo este proceso por cada caso de uso obtendremos un catálogo con las pruebas del sistema necesarias para garantizar la calidad del sistema desarrollado.

4.4. Generación del conjunto de pruebas aplicando UML-Based Statistical Test Case Generation.

El primer paso para aplicar esta propuesta es refinar el caso de uso detallando el camino principal de ejecución, caminos alternativos, combinaciones con otros casos de uso, precondiciones y postcondicones.

Todos estos elementos ya los tenemos recogidos en el caso de uso del apartado 4.1, excepto relaciones con otros casos de uso que no estamos teniendo en cuenta para este ejemplo, por lo que podemos pasar al siguiente paso.

El segundo paso es transformar el caso de uso en un diagrama de estados. El diagrama obtenido es idéntico al diagrama de estados obtenido en el ejemplo de aplicación de la propuesta SCENT. Por motivos de comodidad se reproduce a continuación:

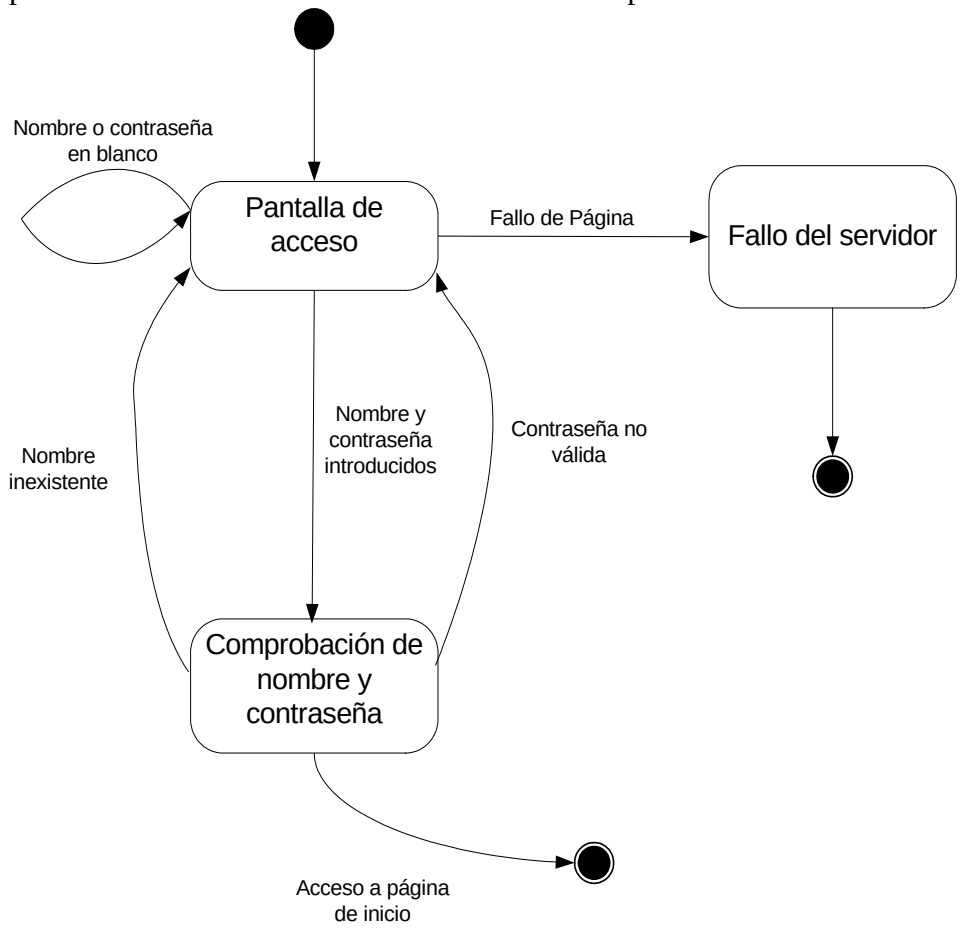

Ilustración 12. Diagrama de estados del caso de uso.

A continuación se transforma el diagrama de estados anterior en un gráfico de uso. En este ejemplo, el diagrama de estados ya incluye todo lo que exige un gráfico de uso, por lo que no es necesario realizar ninguna modificación sobre el diagrama de estados para transformarlo en gráfico de uso.

El método principal para determinar la probabilidad de cada transición es estudiando los usos en el sistema. En este ejemplo se da la peculiaridad de que solo existe un único uso del sistema posible y todos los flujos alternativos de ejecución existen a consecuencia de errores que puedan surgir. Por lo tanto hemos determinado las probabilidades basándonos en nuestra propia experiencia. Las probabilidades asignadas a cada transición se muestran en la siguiente tabla. Estas probabilidades se recogen en la tabla 16.

Símbolo	Descripción	Probabilidad
FP	Fallo de página	0.05
I	Nombre y contraseña introducidos	0.85
B	Nombre o contraseña en blanco.	0.1
NI	Nombre incorrecto.	0.1
NC	Contraseña incorrecta	0.1
A	Acceso a página de inicio	0.8

Tabla 16. Probabilidades de cada transición.

A continuación, con la información de la tabla anterior, elaboramos el modelo de uso añadiendo las probabilidades a cada una de las transiciones del diagrama de estados. Este diagrama se muestra en la ilustración 13.

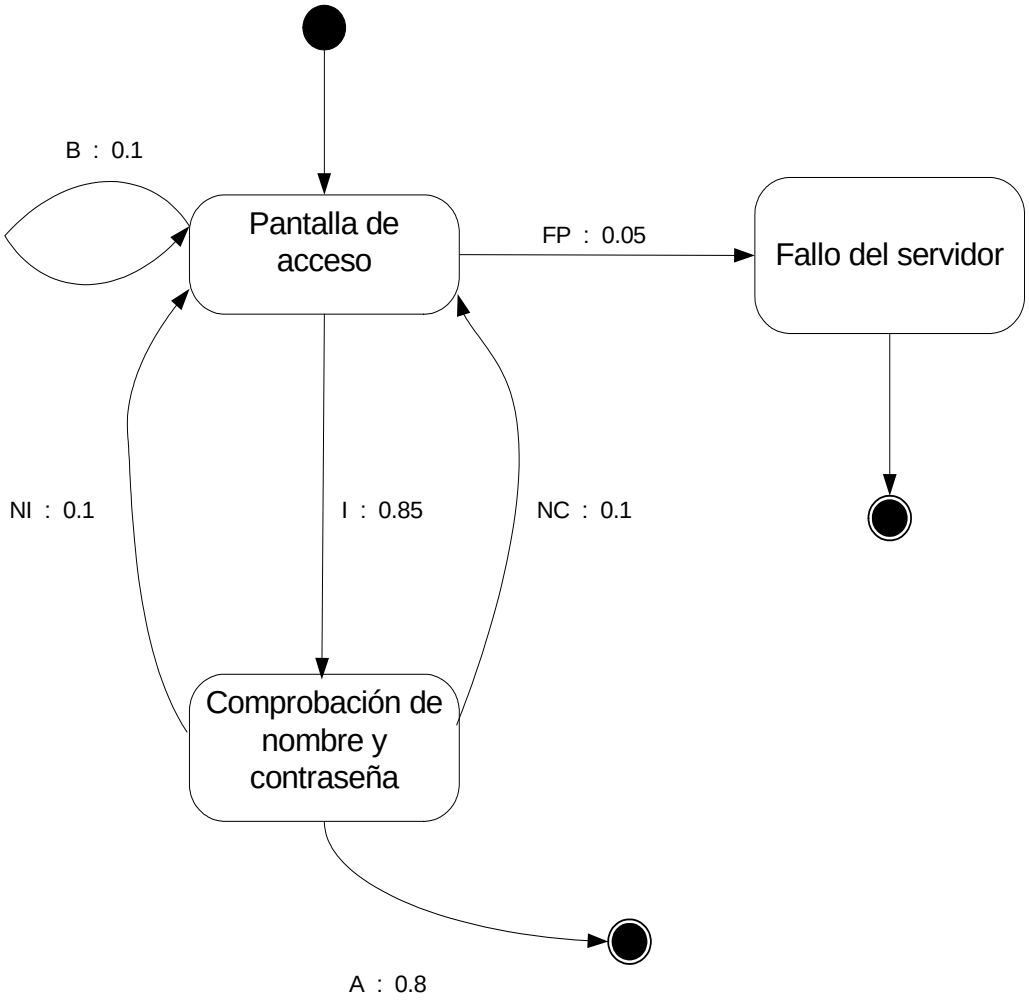

Ilustración 13. modelo de uso.

Este modelo de uso es la base para pruebas aleatorias, las cuales deciden que camino de ejecución seguirán en función de las probabilidades de cada transición.

El único requisito básico para un conjunto de pruebas aleatorias es que contemple el arco mínimo de ejecución, es decir, una secuencia de transiciones que recorran completamente el modelo en el menor número de pasos. En nuestro ejemplo esta secuencia es el camino en el que un actor introduce un nombre de usuario y clave corrector y accede a su página de inicio.

Después de que el camino mínimo sea verificado, las pruebas aleatorias se generan. Cada caso de prueba es un camino aleatorio que recorre el modelo de uso, y que elige la transición que toma en cada estado que atraviesa, dependiendo de las probabilidades de cada transición. Por ejemplo, si ejecutáramos 1.000 pruebas aleatorias sobre el modelo de uso, aproximadamente 50 (el 0,05 o el 5 %) deberían terminar en fallo de página.

4.5. Generación del conjunto de pruebas aplicando Use Case Path Analisys.

El primer paso es dibujar el diagrama de flujo a partir de la secuencia normal de ejecución y las excepciones posibles del caso de uso. Este diagrama de flujo se muestra en la ilustración 5 en el punto 2.5. Por comodidad lo volvemos a reproducir a continuación, en la ilustración 14, incluyendo además un número único para cada rama, lo que nos facilitará la tarea a la hora de identificar los caminos..

La numeración de las ramas no corresponde con la numeración de las secuencia de ejecución o excepciones del caso de uso, ya que en una misma situación, por ejemplo la 4, pueden surgir varias excepciones cada una con su rama de ejecución propia.

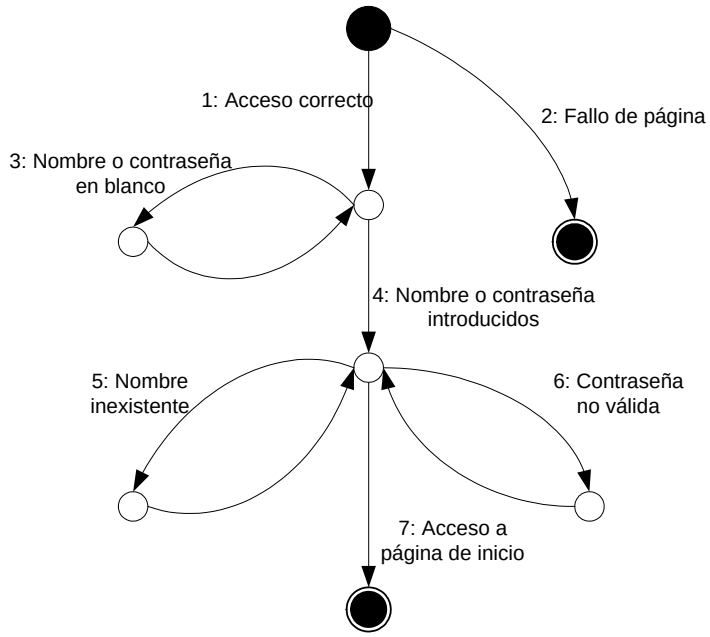

Ilustración 14 Diagrama de flujo del caso de uso.

A continuación se identifican todos los posibles caminos del diagrama de flujo, anotando por cada camino su descripción. En la tabla 17 se muestran todos los caminos identificados a partir del diagrama de flujo anterior.

Id	Nombre	Descripción
1	2	Fallo de página por problemas del servidor.
2	1, 4, 7	Acceso correcto al sistema.
3	1, 3, 4, 7	Acceso correcto al sistema después de introducir un nombre o contraseña en blanco.
4	1, 4, 5, 7	Acceso correcto al sistema después de introducir un nombre inexistente.
5	1, 4, 6, 7	Acceso correcto al sistema después de introducir una contraseña no válida.
6	1, 4, 5, 6, 7	Acceso correcto al sistema después de introducir un nombre inexistente y una contraseña no válida.
7	1, 3, 4, 5, 6, 7	Acceso correcto al sistema después de introducir un nombre o contraseña en blanco, un nombre inexistente y una contraseña no válida.

Tabla 17. Caminos posibles de ejecución.

Existen más caminos posibles, como por ejemplo el camino 1, 3, 4, 5, 7, pero son repeticiones de los caminos ya mostrados en la tabla 13.

A continuación se procede a analizar y baremar todos los caminos identificados. Para la baremación utilizaremos los dos factores indicados por la propuestas: la frecuencia y el factor crítico, sin embargo cambiaremos la escala de puntuación. Los valores posibles para ambos factores, en vez de estar comprendidos entre 1 y 10, estarán comprendidos dentro del rango de 1 a 5, ambos incluidos.

Para el factor de frecuencia un valor de 1 significa un camino que se ejecuta con una frecuencia muy alta, en torno al 90% - 95%, un valor de 3 significa un camino que se ejecuta aproximadamente en la mitad de los recorridos del caso de uso, en torno al 50%, y un valor de 5 significa un camino que se ejecuta con una frecuencia muy baja, en torno al 5% - 10%.

Para el factor crítico un valor de 1 significa que un fallo en este camino de ejecución puede ser recuperado por el sistema con un gasto mínimo de recursos y sin comprometer su información, un valor de 3 significa que un fallo en este camino es importante para el sistema, pero puede seguir trabajando, y un valor de 5 significa que un fallo en este camino es crítico para el sistema, o que puede provocar la caída total del mismo.

Una vez puntuados estos factores para cada camino, se obtiene el factor de cama camino según la fórmula de la tabla 7 (Valoración = Frecuencia + Factor crítico). A continuación, en la tabla 18, se muestra las puntuaciones de cada atributo y el valor del factor, o valoración, de cada camino identificado.

Id	Nombre	Frecuencia	Factor crítico	Valoración
1	2	1	1	2
2	1, 4, 7	5	5	10
3	1, 3, 4, 7	2	5	7
4	1, 4, 5, 7	3	5	8
5	1, 4, 6, 7	4	5	9
6	1, 4, 5, 6, 7	2	5	7
7	1, 3, 4, 5, 6, 7	1	5	6

Tabla 18. Atributos y factor para cada camino.

A continuación se ordenan los caminos en función de su valoración. Los caminos ordenador de mayor a menos valoración se muestran en la tabla 19.

Id	Nombre	Frecuencia	Factor crítico	Valoración
2	1, 4, 7	5	5	10
5	1, 4, 6, 7	4	5	9
4	1, 4, 5, 7	3	5	8
3	1, 3, 4, 7	2	5	7
6	1, 4, 5, 6, 7	2	5	7
7	1, 3, 4, 5, 6, 7	1	5	6
1	2	1	1	2

Tabla 19. Caminos ordenador según su valoración.

El camino principal, que en este ejemplo es el camino que recorre las ramas 1, 4 y 7, generalmente es el camino más importante, por lo tanto, como regla general, es el camino cuyo factor de 5 es el mayor de todos los caminos identificados para un caso de uso. En el ejemplo concreto que hemos seleccionado, todas las ramas alternativas, salvo la rama 2, vuelven al camino principal, por lo tanto cualquier fallo en estas ramas alternativas impide la consecución del camino principal. Por este motivo el factor crítico de estas ramas alternativas coincide con el factor crítico de la rama o camino principal.

El siguiente paso es la selección de los caminos de los que deben derivarse casos de prueba, o que deben convertirse en casos de prueba. Para ello se seleccionan los caminos con una valoración más alta teniendo en cuenta que siempre ha de seleccionarse el camino principal de ejecución y que es necesario que todas las ramas sean comprobadas por un caso de prueba como mínimo. La lista de caminos seleccionados se muestra en la tabla 20.

Id	Nombre	Frecuencia	Factor crítico	Valoración	Descripción
2	1, 4, 7	5	5	10	Acceso correcto al sistema.
5	1, 4, 6, 7	4	5	9	Acceso correcto al sistema después de introducir una contraseña no válida.
4	1, 4, 5, 7	3	5	8	Acceso correcto al sistema después de introducir un nombre inexistente.
3	1, 3, 4, 7	2	5	7	Acceso correcto al sistema después de introducir una contraseña no válida.
1	2	1	1	2	Fallo de página por problemas del servidor.

Tabla 20. Caminos seleccionados para derivarlos en casos de prueba.

Hemos elegido todos los casos de prueba menos el caso con id 6 y el caso con id 7, ya que todas las ramas de estos dos caminos ya se comprueban mediante los casos de prueba de los caminos con id 2, id 3, id 4 e id 5. El camino con id 1 también es elegido para derivar casos de prueba a pesar de tener la valoración más baja de entre todos los caminos, ya que, como hemos mencionado en el párrafo anterior, es necesario que todos los caminos se comprueben por, al menos, un caso de prueba.

A continuación elaboramos un conjunto de pruebas que verifiquen los caminos seleccionamos podemos garantizar que se comprueba adecuadamente el caso de prueba.

Para generar estos casos de prueba se elabora, por cada caso, un conjunto de escenarios de prueba, los cuales consisten en una tabla con datos válidos, que hagan pasar con éxito el caso de prueba, y datos inválidos, que hagan fallar el caso de prueba.

A continuación, en la tabla 21 mostramos un conjunto de datos de prueba para el camino Id 6. En cursiva se muestran los valores válidos.

Atributos	*1*	*2*	*3*
Nombre:	Joh *John*	Yane *Jane*	Peterr *Peter*
Contraseña:	MyK *Dough*	dough *Dough*	Otoole *Parker*
Notas:	1 nombre y contraseñas incorrectos y 1 correctos. Sustituir nombre y clave incorrecta por las correctas.	1 nombre y contraseñas incorrectos y 1 correctos. Sustituir nombre y clave incorrecta por las correctas.	1 nombre y contraseñas incorrectos y 1 correctos. Sustituir nombre y clave incorrecta por las correctas.
Resultado esperado:	Acceso a la página de inicio del usuario John.	Acceso a la página de inicio de la usuario Jane.	Acceso a la página de inicio del usuario Peter.

Tabla 21. Datos de prueba.

5. Análisis y comparación de las propuestas descritas.

5.1. Puntos en comunes y comparación global.

En las propuestas estudiadas se han extraído una serie de puntos comunes que se detallan a continuación:

1. El objetivo de todas las propuestas es obtener un conjunto completo de pruebas del sistema que permitan garantizar que el sistema software cumple con la especificación funcional dada, lo cual permite asegurar su calidad.
2. Todas parten de los requisitos funcionales del sistema y todas permiten comenzar a desarrollar los casos de prueba del sistema en cuanto se comience a disponer los requisitos funcionales.
3. Todas usan el análisis de los caminos posibles, bien mediante la descripción textual de los pasos del escenario o caso de uso, o mediante diagramas de estado.
4. Los requisitos funcionales no tienen que cumplir de principio ningún requisito formal. A partir de una breve descripción en lenguaje natural ya se puede comenzar a trabajar.
5. La derivación de pruebas del sistema a partir de los requisitos funcionales se realiza de manera automática y sistemática. Todas las propuestas son susceptibles de automatización, algunas más que otras, mediante herramientas software.
6. La aplicación de estas propuestas a los requisitos funcionales ayuda a validarlos, comprobando si son correctos y completos en las primeras fases de desarrollo.

En la tabla 22 se muestra una comparativa entre las distintas propuestas basada en sus aspectos más relevantes. En la tabla 23 se describen las características que se han tenido en consideración para comparar las propuestas.

	SCENT	Generación de casos de prueba a partir de CU.	UML-Based Statistical Test Case Generation	AGEDIS	Use Case Path Analysis
Nueva notación	Sí *(1)*	No	No	Sí *(2)*	Sí
Automatización completa	No	No	No	Sí	No
Casos prácticos.	Sí	No	No	Sí	Sí
Tipo de pruebas	Fallos	Fallos	Fiabilidad	*(5)*	Fallos
Nivel de automatización.	Generación	Generación	Generación	Ejecución	Generación
Uso de estándares	Sí	Sí	Sí	Sí	Sí
Automatización por herramientas	Sí	No	Sí	Sí	Sí
Herramientas de soporte	No	No	No	Sí	No
Dificultad de implantación	Media	Baja	Media	Alta	Baja
Ejemplos	Sí	Sí	Sí	Sí	Sí
Número de pasos	16+3 *(3)*	3	5	6	4+1 *(4)*
Análisis y selección de casos de prueba	Sí	No	No	Sí	Sí

Tabla 22. Tabla comparativa de las propuestas.

(1) – Diagramas de dependencias de los escenarios.

(2) – Lenguaje IF para modelar el sistema.

(3) – 16 pasos para obtener los escenarios y 3 pasos para derivar casos de prueba a partir de ellos.

(4) – 4 pasos para obtener el conjunto de caminos y un paso adicional para convertirlos en casos de prueba

(5) – Los objetivos de las pruebas pueden ser encontrar fallos, realizar pruebas de carga, de fiabilidad, etc..

Nueva notación:	Indica si la propuesta propone una notación o un conjunto de diagramas propio.
Automatización completa	Indica si las propuestas ofrecen un sistema de automatización total o dejan partes que deben realizarse a mano.
Casos prácticos	Indica si existen referencias de proyectos reales en los que la propuesta se haya llevado a la práctica.
Tipo de pruebas	Indica si los casos de uso generados están orientados a la búsqueda de fallos del sistema o a pruebas de fiabilidad.
Nivel de automatización.	Indica si la propuesta contempla también la automatización de la ejecución de pruebas o solo la automatización de la generación de casos de prueba.
Uso de estándares	Indica si la propuesta se basa en herramientas y diagramas ampliamente usados, como UML.
Automatización por herramientas	Indica si todo el proceso es susceptible de ser automatizado mediante herramientas software. La propuesta generación de casos de prueba a partir de los casos de uso trabaja siempre lenguaje natural, por lo que no es susceptible de automatización, aunque sí se pueden desarrollar herramientas que gestionen el proceso.
Herramientas de soporte	Indica si actualmente existen herramientas que den soporte a la metodología o automaticen sus pasos.
Dificultad de implantación	Una dificultad baja indica una propuesta muy sencilla de aplicar y para la que apenas se requiere preparación previa. Una dificultad media indica que existen elementos en la propuesta, nueva notación o algún proceso con alto grado de dificultad, que requiere preparación previa. Una dificultad alta indica que no se puede aplicar una propuesta sin un estudio detallado de todos sus elementos. En el siguiente apartado se detalla, para cada propuesta, el motivo de la dificultad de implantación asignada.
Ejemplos	Indica si la propuesta incluye ejemplos de los distintos pasos.
Número de pasos	Índica el número de pasos a dar para obtener un conjunto de pruebas del sistema.
Análisis y selección de casos de prueba	Indica si la propuesta describe como analizar y seleccionar solo aquellos casos de uso o caminos de ejecución relevantes para convertirlos en casos de prueba.

Tabla 23. Descripción de las características evaluadas.

5.2. Principales puntos fuertes y puntos débiles de las propuestas estudiadas.

A continuación se resumen los principales puntos fuertes y débiles de cada una de las propuestas analizadas. También, tomando como base los factores analizados en las tablas anteriores, se indica cual es el nivel de dificultad de implantación de cada una de las propuestas. En la ilustración 15 se resume el nivel de dificultad de todas las propuestas analizadas.

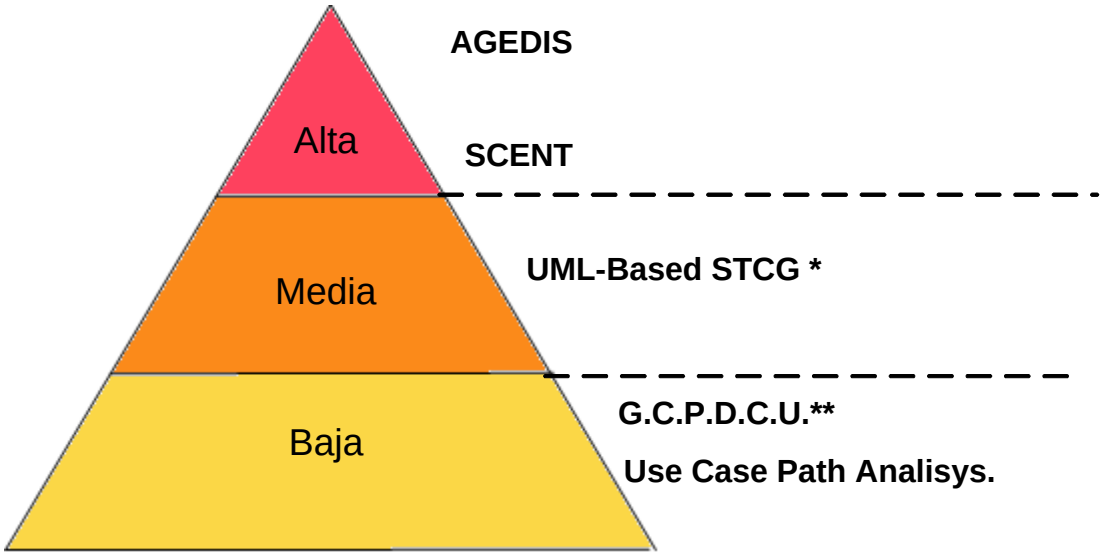

Ilustración 15 Dificultad de implantación de las propuestas analizadas.

* UML-Based Statistical Test Case Generation.
** Generación De Casos De Prueba A Partir De Casos De Uso.

5.2.1. SCENT.

Su principal punto fuerte es que ofrece un método muy completo para crear y organizar escenarios de uso. Además incluye dos referencias a proyectos reales donde se ha aplicado con éxito.

Su principal punto débil es que es necesario realizar un trabajo muy extenso, 16 pasos, con los escenarios antes de poder generar casos de prueba.

La dificultad de implantación es media porque incluye un nuevo modelo de diagrama, orientado a representar las dependencias que los requisitos funcionales tienen entre sí, que es necesario aprender.

5.2.2. Generación de casos de prueba a partir de casos de uso.

Su principal punto fuerte es también su principal punto débil. El trabajar directamente con casos de uso en lenguaje natural, en vez de traducirlos a una representación formal, permite obtener rápidamente casos de prueba, pero dificulta que este proceso pueda sistematizarse y automatizarse mediante herramientas. Tampoco incluye ninguna referencia a ningún proyecto donde se haya aplicado.

La dificultad de implantación es baja, porque no requiere ningún formalismo sino que se basa directamente en el lenguaje natural y solo necesita tres pasos para obtener un conjunto de casos de prueba a partir de un caso de uso.

5.2.3. UML-Based Statistical Test Case Generation.

Su principal punto fuerte es que es la única propuesta que permite derivar casos de prueba de fiabilidad a partir de los requisitos funcionales. Además, es posible aplicar esta propuesta para obtener casos de prueba de fiabilidad conjuntamente con alguna de las otras propuestas analizadas para obtener, además, casos de prueba que permitan encontrar fallos en el sistema.

Como puntos débiles, no se incluyen referencias de ningún caso práctico o proyecto donde se haya aplicado. Tampoco ofrece un modelo general para determinar la probabilidad de cada camino de ejecución, sino que para caso ha de ser calculada estudiando los diagramas de estados y las especificaciones suministradas por el cliente. Por este motivo la dificultad de implantación es media.

5.2.4. AGEDIS.

Los puntos fuertes más destacables se concretan en que es la propuesta más completa, abarcando tanto la generación como la ejecución y automatiza de los casos de prueba. Además es posible automatizar todos los pasos con herramientas de libre acceso para fines no lucrativos y proporciona referencias a varios proyectos reales donde se ha aplicado con éxito.

Como puntos débiles, el adoptar herramientas poco conocidas, como el lenguaje de modelado IF, en vez de usar soluciones más extendidas como OCL o un subconjunto de Java; el no ser adecuado para todo tipo de proyectos sino solo en los que prima el control de flujo por encima de la transformación de información, y que las empresas tienen que realizar una inversión elevada para poder adquirir el conjunto de herramientas que soportan esta propuesta. También es un punto débil su dificultad de implantación, dado que AGEDIS cuenta con un conjunto de formalismos, documentación y herramientas propias que es necesario conocer antes de poder aplicarlo.

AGEDIS es la única propuesta que obtiene el aprobado en el apartado de automatización completa pues ofrece herramientas para la generación y realización de pruebas, quedándose el resto de propuestas solo en la generación de pruebas.

5.2.5. Use Case Path Analisys.

Esta propuesta es muy similar a la propuesta Generación de Casos de Prueba a Partir de Casos de Uso[1]. Sin embargo incluye dos nuevos elementos importantes. En primer lugar, aunque el proceso es muy similar en ambas propuestas, Use Case Path Analisys incluye como primer paso la traducción del caso de uso de lenguaje natural a una representación gráfica formal, aunque no estándar, lo que facilita su automatización por herramientas.

En segundo lugar propone una manera para poder determinar que caminos de ejecución son más frecuentes y críticos y eliminar los caminos que no aportan nada, con lo que se reduce el conjunto de casos de prueba sin comprometer la calidad de las pruebas.

Como puntos débiles: la notación que propone para diagramas de flujo es muy simple y poco práctica para casos de uso con muchos posibles caminos. Tampoco da ninguna información sobre como crear los casos de uso una vez identificados los caminos más importantes ni como obtener escenarios de prueba adecuados.

Su dificultad de implantación es baja ya que, aunque incorpora un formalismo propio, diagrama de flujo, estos diagramas no son difíciles de elaborar y peden ser perfectamente sustituidos por diagramas de estados UML.

7. Conclusiones y futuros trabajos.

7.1. Conclusiones.

La planificación de las pruebas del sistema en las primeras fases del desarrollo ayudan a validar los requisitos funcionales. Esta planificación debe empezar tan pronto como se comience a disponer de los requisitos funcionales. La obtención de los casos de prueba del sistema puede sistematizarte e integrarse en el proceso de desarrollo, en paralelo a la fase de elicitación de requisitos.

Para la obtención de un conjunto de casos de prueba para las pruebas del sistema a partir de un conjunto incompleto de requisitos hemos descrito cinco propuestas que nos permiten obtener casos de prueba de forma sistemática para realizar las pruebas de sistema en las primeras fases del ciclo de vida del sistema software. Existen bastantes diferencias entre las propuestas, lo cual nos permite elegir una de acuerdo con la inversión en tiempo y recursos disponible. Así, por ejemplo, podremos empezar con un método sencillo para pequeños proyectos, como [1] u [8], e ir gradualmente evolucionando a propuestas más completas y complejas.

De todas las propuestas estudiadas, AGEDIS [5], [6] es la más moderna y completa pero no es la solución definitiva. De hecho ninguna de las propuestas es definitiva, todas tienen alguna ventaja sobre las demás y algunos puntos débiles.

Este estudio también pretende servir también de guía a la hora de elegir una propuesta para aplicarla a un contexto industrial, ya que se han detallado las características tanto de propuestas ligeras de rápida implantación sin apenas costes, como la propuesta más actual y compleja.

7.2. Futuras líneas de investigación.

Una línea de investigación abierta a partir de este informe es el desarrollo de una nueva propuesta metodológica para la derivación de casos de prueba, que tome como punto de partida todos los puntos comunes de las propuestas estudiadas, agrupe todos sus puntos fuertes y corrija todos sus puntos débiles y tome en consideración aspectos no contemplados en las propuestas estudiadas, como la estimación el número de casos de uso necesario, y evaluar los casos de uso obtenidos descartando los que se solapen con otros.

Otra línea de investigación abierta es realizar un estudio el proceso de implantación de las metodologías de generación de casos de prueba en entornos industriales. La aplicación de una metodología de prueba a un contexto industrial impone requerimientos adicionales y restricciones no existentes en una aproximación puramente académica, como el retorno de la inversión. También es necesario estudiar el proceso de integración de estas propuestas metodológicas dentro de los procesos de desarrollo existentes en la industria.

8. Referencias.

[1] Heumann , Jim, 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals.

[2] Johannes Ryser, Martin Glinz 2003. Scent: A Method Employing Scenarios to Systematically Derive Test Cases for System Test. Technical Report 2000/03, Institut für Informatik, Universität Zürich.

[3] Johannes Ryser, Martin Glinz 2003. A Practical Approach to Validating and Testing Software Systems Using Scenarios Quality Week Europe QWE'99 in Brussels, Institut für Informatik, Universität Zürich.

[4] Matthias Riebisch, Ilka Philippow, Marco Götze Ilmenau. UML-Based Statistical Test Case Generation. Technical University, Ilmenau, Germany.

[5] Hartman , Alan 2004 AGEDIS Final Project Report AGEDIS Consortium Internal Report. http://www.agedis.de/

[6] Cavarra, Alessandra, Davies, Jim 2004 Modelling Language Specification. AGEDIS Consortium Internal Report. http://www.agedis.de/

[7] Jalote, Pankaj, 2002. Software Project Management in Practice. Addison Wesley. USA.

[8] Ahlowalia, Naresh. 2002. Testing From Use Cases Using Path Analysis Technique. International Conference On Software Testing Analysis & Review.

[9] R. Itschner, C. Pommerell, M. Rutishauser, "GLASS: Remote Monitoring of Embedded Systems in Power Engineering," IEEE Internet Computing, vol. 2, # 3, pp. 46-52, 1998.

[10] IBM Software Testing & Verification technologies. http://www.haifa.il.ibm.com/software_testing.html

[11] Craggs I., Sardis M., and Heuillard T., AGEDIS Case Studies: Model-based testing in industry. Proc. 1st European Conference on Model Driven Software Engineering, 106-117. imbus AG December 2003.

[12] Robinson H., Obstacles and opportunities for model-based testing in an industrial software environment. Proc. 1st European Conference on Model Driven Software Engineering, 118-127. imbus AG December 2003.

[13] Jalote, Pankaj, 2002. Software Project Management in Practice. *Addison Wesley.*

[14] A. Bertolino, E. Marchetti,H. Muccini. 2004. Introducing a Reasonably Complete and Coherent Approach for Model-based. *Electronic Notes in Theoretical Computer Science.*

[15] Métrica 3. Ministerio de Administraciones Públicas. http://www.map.es/csi