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

Generación de pruebas de sistema a partir de la especificación funcional

Universidad de Sevilla

Lenguajes y Sistemas Informáticos

España

Javier Jesús Gutiérrez

Tutores:

Doctor D. Manuel Mejías Risoto.

Doctora Dña. María José Escalona Cuaresma.

En Sevilla, Julio de 2.005

Presentación del informe de investigación.

El aumento de la complejidad de los sistemas software pone de manifiesto la necesidad de garantizar la calidad de dichos sistemas. En el contexto de este trabajo definimos calidad como la adecuación del software a sus requisitos.

En este informe de investigación proponemos el uso de las pruebas de sistema como herramienta para verificar el cumplimiento de la especificación funcional de un sistema y asegurar su calidad. En las secciones 1 y 2 se definen las pruebas de sistema y se exponen las ventajas de contar con propuestas que indiquen cómo obtener un conjunto de pruebas adecuadas y permitan automatizar este proceso.

Loa puntos principales de este informe de investigación, secciones 3, 4 y 5, describen un análisis comparativo de las propuestas existentes para guiar la generación de casos de prueba del sistema. Las conclusiones de este análisis, recogidas en las secciones 6 y 7, permiten establecer el nivel de detalle y madurez de estas propuestas, así como sentar las bases para un futuro trabajo de investigación.

Por último, la sección 8 resume las publicaciones, congresos, institutos y grupos de investigación más relevantes en la investigación sobre generación de casos de prueba

Índice:

Índice de tablas:

Índice de ilustraciones:

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 dentro del desarrollo de cualquier sistema informático.

El proceso de prueba del software puede definirse como la verificación dinámica del comportamiento del software a partir de un conjunto finito de casos de prueba [Swebok04].

Un aspecto fundamental del proceso de prueba es evaluar la satisfacción de la especificación funcional del sistema o requisitos por parte del sistema construido. Para garantizar el nivel de calidad del sistema construido es necesario verificar la correcta y completa implantación de los requisitos establecidos en las etapas iniciales del desarrollo [Jalote02]. Una herramienta adecuada para efectuar esta validación son las pruebas del sistema. Una definición de prueba del sistema se muestra en la tabla 1.

Tabla 1. Definición de prueba del sistema.

Las pruebas del sistema tienen como objetivo verificar la funcionalidad del sistema a través de sus interfaces externas comprobando que dicha funcionalidad sea la esperada en función de los requisitos del sistema [Swebok04].

A lo largo de esta sección se describen los conceptos utilizados en este trabajo de investigación. En el punto 1.1 se estudian las características comunes a todas las pruebas. Estas características también aparecerán en las pruebas de sistema. En el punto 1.2 se describe un posible proceso de prueba del software y dónde encajan las pruebas de sistema dentro de este proceso.

1.1. Definición de las pruebas software.

De una manera general es posible definir una prueba como la ejecución de una aplicación, o un trozo de código, para identificar uno o varios errores [Pressman04]. Se

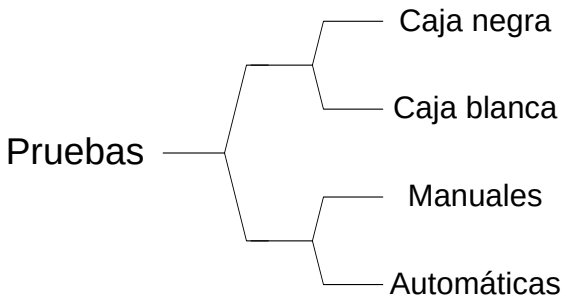
define error como un resultado distinto del resultado esperado [Pressman04]. Algunas características comunes a todas las pruebas software se recogen en la tabla 2 [Binder00].

Tabla 2. Características comunes a las pruebas y al proceso de prueba.

1. Para probar necesitamos código que se pueda ejecutar.
2. Para probar necesitamos saber cual es el resultado esperado.
3. Una prueba es mejor que otra cuando encuentra más errores.
4. Es imposible probar una aplicación al 100%.
5. Las pruebas no deben dejarse para el final.

Es posible clasificar las pruebas en dos grandes grupos: pruebas de caja negra y caja blanca y pruebas manuales y automáticas [Pressman04], tal y como se muestra en la ilustración 1 y se describe en los párrafos siguientes.

Ilustración 1. Tipos de pruebas.



Las pruebas de caja blanca tienen como objetivo recorrer la estructura del código comprobando la ejecución de todos los posibles caminos de ejecución. Las pruebas de caja negra, en cambio, se centran en los requisitos o resultados del sistema software. Al contrario que en las pruebas de caja blanca, en las pruebas de caja negra no interesa la estructura interna del código, sino que consideran al sistema como una caja negra, de ahí su nombre. A dicha caja se le suministran una serie de valores de entrada y se verificará que las salidas sean las esperadas.

Las pruebas automáticas son aquellas realizadas por un programa o herramienta que prueba el sistema sin necesidad de la interacción de una persona. La herramienta suministra una serie de valores de prueba, o acciones de prueba al sistema y verifica los resultados devueltos por el sistema con los resultados esperados. Al final del proceso de prueba se emite un informe con los resultados de las mismas. Cuando se desea repetir el

proceso de prueba, generalmente después de una modificación en el mismo, solo es necesario volver a ejecutar la herramienta y emitir un nuevo informe.

Las pruebas manuales, en cambio, son aquellas pruebas realizadas por una o más personas que interactúan directamente con el sistema. Estas personas verifican si los resultados obtenidos son válidos o no. Cuando se desea repetir el proceso es necesario que la persona o grupo repita las interacciones y vuelvan a verificar todos los resultados obtenidos.

Las pruebas automáticas suelen ser más deseables que las manuales ya que evitan el factor humano. La incidencia del factor humano en el proceso de prueba se estudia con más detalle en la sección 2.

Las pruebas del sistema descritas en este trabajo son pruebas de caja negra automáticas.

1.2. Un proceso de prueba del software.

En este punto se describe un proceso global de prueba del software y dónde encajan las pruebas del sistema dentro de este proceso.

Son muchas las clasificaciones de los tipos de prueba que se pueden realizar. Una clasificación posible es la propuesta por Métrica 3 [Métrica3] y que se muestra en la tabla 3.

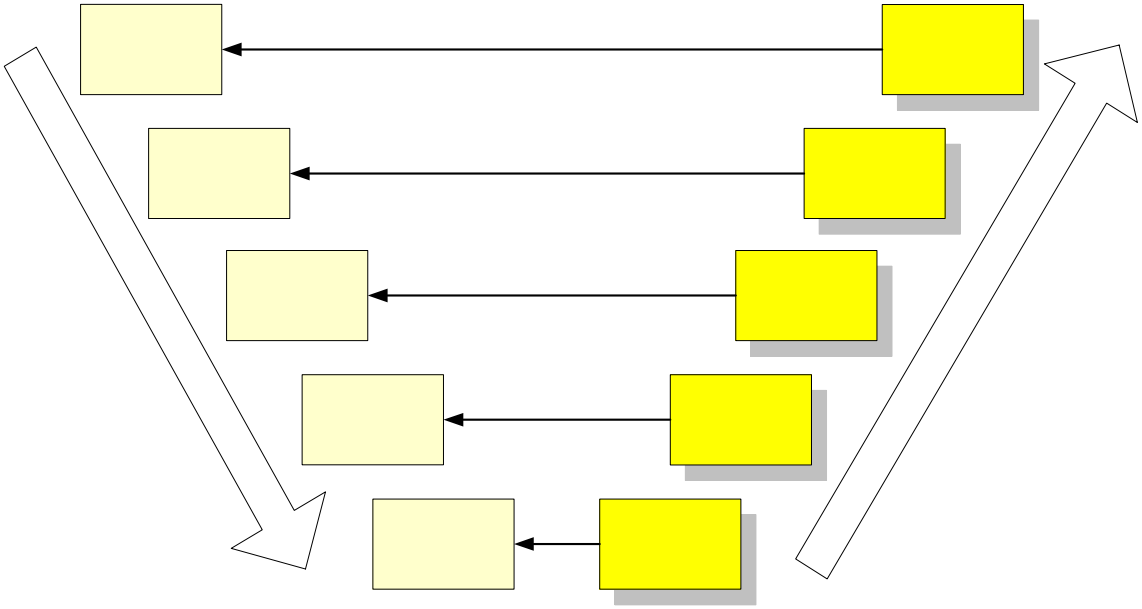
Tabla 3. Descripción de los tipos de prueba del software.

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	Después de realizar	El objetivo es comprobar que los cambios sobre un componente del sistema, no generan errores

Regresión.	modificaciones al sistema.	adicionales en otros componentes no modificados.

En la ilustración 2 se muestra sobre qué elementos del sistema se aplica el conjunto de pruebas de la tabla 3.

Ilustración 2. Proceso de prueba del software.



Para que el proceso de prueba del sistema sea eficaz debe estar integrado dentro del propio proceso de desarrollo. Como cualquier otra fase de dicho proceso, el proceso de prueba 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.

En la siguiente sección se profundizarán en las pruebas del sistema.

2. Planteamiento del problema.

En la sección anterior se ha estudiado un proceso de prueba del software. También se ha visto la utilidad de las pruebas del sistema para garantizar la calidad del producto final [Jalote02]. En esta sección se expondrá la necesidad de contar con una metodología para la generación de pruebas del sistema, las ventajas que aporta y los problemas que resuelve.

Para alcanzar su máxima eficiencia, un conjunto de pruebas del sistema debe cumplir las características recogidas en la tabla 4 [Swebok04].

Tabla 4. Características de los conjuntos de pruebas.

• Deben ser finitas. • Deben ser mínimas (*). • Deben ser fácilmente repetibles (**).

(*) – El mínimo conjunto de pruebas que permita alcanzar el nivel de prueba establecido.

(**) – Rara vez un conjunto de pruebas se ejecuta una única vez. Si un conjunto de pruebas descubre errores, es normal ejecutarlas una segunda vez para verificar que dichos errores han sido corregidos.

A continuación se exponen los distintos problemas que surgen al construir manualmente un conjunto de pruebas del sistema que cumplan las características recogidas en la tabla 3. Estos problemas justificarán la necesidad de contar con procesos de generación de pruebas de sistema que puedan ser realizados de manera sistemática y automática.

2.1. Problemas de las pruebas manuales.

Las pruebas manuales han sido definidas en la sección 1.1. Una definición de verificación y validación [IEEE90] se muestra en la tabla 5.

Tabla 5. Definición de verificación y validación.

El proceso de determinar cuándo los requisitos de un sistema son completos y correctos, cuándo los productos de cada una de las fases de desarrollo satisfacen los requisitos o condiciones impuestas por las fases anteriores y cuándo el sistema final o componente cumple los requisitos especificados.

Las primeras definiciones de pruebas no han variado hasta nuestros días. Ya en el año 1979 [Myer79] definía el proceso de prueba del software como la búsqueda de fallos mediante la ejecución del sistema o de partes del mismo en un entorno y con valores de entrada bien definidos.

Sin embargo, este proceso aplicado a las pruebas del sistema puede resultar tedioso y repetitivo. Si estas pruebas se desarrollan manualmente por un conjunto de técnicos, el proceso de prueba consistiría en introducir manualmente diversas combinaciones de valores de prueba y comparar los resultados con los resultados esperados. Estos técnicos deben repetir el mismo conjunto de pruebas cada vez que se modifique el sistema. Este proceso provoca fatiga y falta de atención, lo cual repercute en su calidad y nivel de detección de pruebas. Además requiere el gasto de los recursos necesarios para tener a una persona, o grupo de personas repitiendo las pruebas [Robinson00].

Los informes de fallos son elaborados por las personas en función de sus apreciaciones, por lo que si una persona está cansada o confunde los resultados de distintas pruebas puede dar lugar a informes erróneos.

La idea de que un proceso de desarrollo de software facilita la construcción del sistema y aumenta su calidad es mundialmente aceptada en la actualidad. Esta misma idea puede y debe aplicarse a los procesos de prueba y, en especial, a la identificación, selección y generación de casos de prueba.

2.2. Calidad del conjunto de pruebas.

Las pruebas no son una manera infalible de garantizar que una aplicación no tiene errores. Muy al contrario, una aplicación puede pasar cientos, miles o millones de

pruebas y tener grandes errores. Las pruebas solo garantizan la existencia o no de los errores que buscan. Por tanto, una aplicación que pase todas las pruebas es una aplicación que no tiene ninguno de los errores que dichas pruebas buscan, pero puede tener otros errores.

Por este motivo es importante planificar y diseñar bien las pruebas para buscar el máximo número de errores posibles o, al menos, los más importantes. Para no depender en exclusiva de los conocimientos y experiencia de los ingenieros de prueba, es necesario un proceso metodológico que ofrezca una pauta clara para seleccionar el conjunto de pruebas que abarque un mayor rango de posibles errores. O bien obtener un conjunto de pruebas centrado en localizar errores con unas características concretas, cómo, por ejemplo, el nivel de rendimiento del sistema ante cargas extremas de trabajo.

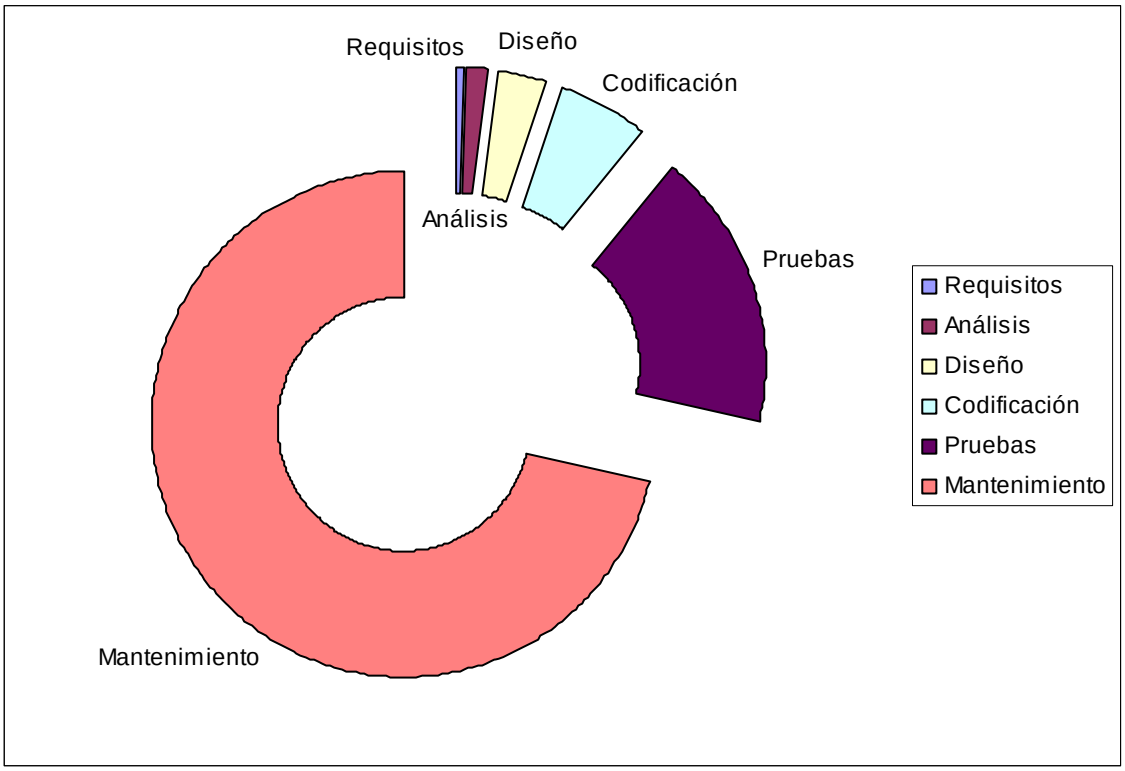
2.3. Costes del proceso de prueba.

Un estudio de 1027 proyectos de sistemas de información revela que solo el 12,7% concluyó satisfactoriamente [Crocker05]. La baja calidad de los requisitos fue considerado el principal motivo de fracaso en el 76% de los proyectos.

Los problemas más comunes en los requisitos son: ambigüedad, imprecisión, inexactitud e inconsistencia [Stokers91]. Asegurar la calidad de los requisitos es una tarea casi imprescindible para culminar un proyecto de desarrollo de software con éxito. Existen varias técnicas ampliamente aceptadas y documentadas para mejorar la calidad de los requisitos, como pueden ser revisiones formales, inspecciones, listas de verificación, ontologías, auditorias, matrices de rastreabilidad, métricas de calidad, etc.

Desde mediados de los años setenta se estima que el coste de corregir un error en un sistema software aumenta a medida que avanza el desarrollo del sistema (ilustración 3). El coste de corregir un error en las últimas etapas se estima entre 50 y 100 veces por encima del coste de corregir en mismo error en las primeras etapas [Boehm75].

Ilustración 3. Coste de corrección de errores en las etapas del desarrollo del software.



Otras estimaciones posteriores [Littlewood87] confirman esta tendencia. Cuanto antes sea identificado y corregido un error, menor será el coste de su corrección [Littlewood87]. En la tabla 6 se muestra el coste relativo de corregir un error, dependiendo de donde sea detectado, en dólares USA.

Tabla 6. Coste relativo en dólares de la corrección de errores (1987).

Fase	**Coste**
Definición	$1
Diseño de alto niveñ	$2
Diseño de bajo nivel	$5
Codificación	$10
Pruebas unitarias	$15
Pruebas de integración	$22
Pruebas del sistema	$50
Producción	$100+

Otros estudios más actuales, como [Mogyorodi01], siguen confirmando esta tendencia. Las estimaciones de [Mogyorodi01] se recogen en la tabla 7.

Tabla 7. Coste de corrección de errores (2001).

Fase	**Porcentaje**
Requisitos	1
Diseño	3-6
Codificación	10
Pruebas unitarias / integración	15-40
Pruebas de sistema / aceptación	30-70
Producción	40-1000

Actualmente, estas estimaciones siguen teniendo plena vigencia y valores similares.

Como se aprecia en la ilustración 3, los costes de corregir errores en la fase de prueba son mucho mayores que los costes de corregir dichos errores en fases tempranas, como la fase de requisitos o de análisis. Adelantar la fase de pruebas a la fase de requisitos o de análisis permite reducir el coste de corrección de errores. Esto debe complementarse con estrategias que permitan, además, disminuir el número de errores potenciales.

Sin embargo, como se ha visto en la sección 1, la realización de una prueba implica la ejecución del código. Para probar es necesario disponer del código del sistema. Por este motivo no es posible adelantar toda la fase de pruebas a etapas tempranas como la fase de requisitos o análisis.

Sin embargo, sí es posible adelantar la generación de casos de prueba a etapas tempranas. La generación de pruebas en la etapa de requisitos o análisis va a permitir encontrar fallos, inconsistencias, omisiones, ambigüedades y sobreespecificaciones en las especificaciones del sistema cuando es más económico corregirlas.

Además, adelantando la generación de pruebas se evita dejar todo el proceso de prueba para final de la construcción, momento donde el tiempo y los recursos disponibles suelen ser insuficientes para realizar este proceso adecuadamente.

2.4. Acotación del problema de búsqueda.

Es imposible probar un sistema al 100% [Binder00]. Por ejemplo, supongamos una función suma que admite como parámetros dos valores de tipo entero y devuelve un valor de tipo entero que es la suma de dichos parámetros. Para probar ese método de manera exhaustiva y garantizar que no va a suceder ningún error con el 100% de exactitud, necesitaríamos probar el método con cada uno de los valores enteros posibles. Como, en teoría, existen infinitos valores enteros obtendríamos infinitos casos de prueba.

Sin embargo, todo microprocesador y sistema operativo manipulan números enteros dentro de un rango de valores concreto. Por ejemplo, en el lenguaje Java existen 2^32 posibles valores enteros [Java04]. Probando todas las posibles combinaciones obtendríamos 2^32 * 2^32 posibles casos de prueba, por lo que el resultado final serían: 18,446.744,073.709,551.616 casos de prueba. Realizando 100 millones de comprobaciones por segundo, se tardarían más de 5800 años. Un valor demasiado alto para ejecutarlos en la práctica.

Muchas de esas pruebas desbordarán el tamaño de un entero, por lo que podemos seleccionar solo aquellas combinaciones de valores que no desborden el valor de un entero y añadir una prueba adicional que sí lo desborde. En este caso tenemos: 4,294.967.296 + 1 casos de prueba.

De una forma sencilla hemos conseguido reducir el número de valores de prueba de infinito a algo más de 4.000 millones, un número mucho más manejable. Sin embargo aún sigue siendo un número demasiado alto.

¿Por qué 4.000 millones de pruebas es un número de pruebas demasiado elevado para utilizarlas en la práctica?.

Podríamos responder que tantas pruebas consumirían mucho tiempo tanto preparándolas como ejecutándolas. Pero esto no es cierto. Los ordenadores cada vez son más rápidos y potentes, por lo que solo sería cuestión de adquirir un ordenador más potente (o alquilarlo) y construir un generador automático de pruebas.

La verdadera razón que responde a la pregunta de porqué 4.000 millones de pruebas es un número demasiado alto e imposible de llevar a la práctica es porque no son necesarias tantas pruebas. Podemos encontrar fácilmente, utilizando el sentido común, un conjunto de no más de 6 o 12 pruebas que prueben todas las distintas

combinaciones posibles. Si con 12 pruebas es posible verificar adecuadamente la función, ¿para qué vamos a realizar 4.000 millones de pruebas?.

Un posible listado de estas pruebas se muestra en la tabla 8.

Tabla 8. Conjunto de pruebas.

Sumando A	**Sumando B**	**Resultado**
Positivo	Positivo	Positivo
Positivo	Negativo	Positivo
Positivo	Negativo	Negativo
Negativo	Positivo	Positivo
Negativo	Positivo	Negativo
Negativo	Negativo	Negativo
Positivo	Positivo	Desbordamiento
Negativo	Negativo	Desbordamiento

¿Por qué hacer 4.000 millones de pruebas cuando solo con 8 es suficiente?. No hay ninguna razón para sobreprobar (overtesting) el sistema. Sin embargo, localizar el conjunto mínimo de pruebas que verifican, casi al 100%, un sistema no es una tarea sencilla. Por este motivo necesitamos técnicas y metodologías que nos guíen en dicho proceso.

Una justificación con más detalle de que es imposible probar al 100% un sistema software puede encontrase en [Goed86].

3. Estudio de la situación actual.

En los apartados de la sección anterior, se han expuesto diversos motivos por los que es necesario contar con una metodología que guíe la generación del conjunto de casos de prueba del sistema. En esta sección se estudian varias propuestas que indican como obtener pruebas de sistema a partir de la especificación funcional del propio sistema.

Actualmente, sólo hemos encontrado un único estudio sobre propuestas de generación de casos de prueba del sistema a partir de las especificaciones del mismo. Este estudio [Quasar03] data de 2.003 y describe 12 propuestas.

Sin embargo, este estudio solo describe las propuestas de manera superficial, limitándose a copiar lo que cada autor dice de su propuesta y sin intentar ponerlas en práctica. También incluye propuestas antiguas, por ejemplo de 1988, claramente desfasadas y difíciles de aplicar a sistemas orientados a objetos o sistemas web. En la realización de este trabajo hemos encontrado propuestas anteriores al año 2.003 (como la propuesta descrita en el punto 3.6) que no han sido incluidas en [Quasar03], por lo que sus resultados pueden no ser correctos. Además, las conclusiones de [Quasar03] estudio son pobres, apenas dos páginas, y superficiales.

Por todos estos motivos consideramos justificado realizar un nuevo estudio comparativo de propuestas de generación de casos de prueba a partir de especificaciones funcionales. En los siguientes apartados se describen cada una de las propuestas analizadas en este trabajo de investigación, de forma mucho más detallada que en [Quasar03] e incluyendo nuevas propuestas más actuales no presentes en dicho informe. Para cada propuesta se indica su punto de partida, los resultados obtenidos, los pasos que la componen y un resumen con sus principales ventajas e inconvenientes.

Las propuestas seleccionadas para este estudio son aquellas que cumplen las siguientes características.

1. Deben comenzar a partir de la especificación funcional del sistema.
2. La especificación debe estar escrita en lenguaje natural.
3. Deben generar pruebas del sistema.
4. Deben ser lo más actuales posibles.
5. Deben poder aplicarse a cualquier tipo de sistema.

De todas las propuestas analizadas, cinco también están presentes en el estudio de Quasar. Estas cinco propuestas están descritas en los puntos 3.3, 3.9, 3.10, 3.11 y 3.12.

Al final de esta sección, en el punto 3.13, se describen brevemente otras propuestas y trabajos estudiados que, por no cumplir estas características, no han sido incluidos en el estudio.

3.1. Automated Test Case Generation from Dynamic Models.

Esta propuesta [Fröhlich00] se puede dividir en dos bloques. En el primer bloque se realiza una traducción de un caso de uso a un diagrama de estados. En el segundo bloque se genera un conjunto de pruebas de sistema a partir de dicho diagrama de estados.

El punto de partida de esta propuesta es un caso de uso en lenguaje natural definido mediante una plantilla. Esta propuesta no ofrece ningún modelo de plantilla a seguir, pero recomienda utilizar las plantillas propuestas en [Cockburn00].

Al final del proceso descrito en esta propuesta se obtiene un conjunto de transiciones posibles sobre un diagrama de estados. Estas transiciones están expresadas mediante operadores. Además, se obtienen las proposiciones iniciales y finales de dicho conjunto. Dicho conjunto deberá implementarse y ejecutarse sobre la aplicación para comprobar si se producen las mismas transiciones y el resultado es el esperado. Un ejemplo de un conjunto de operadores y de transiciones obtenido mediante esta propuesta se muestran en las tablas 9 y 10.

Tabla 9. Operadores, estado inicial y meta del sistema.

Operadores:	(operator a0 (preconds (in initial)) (effects (del in initial) (in main-screen))) (operator a1 (preconds (in main-screen)) (effects (del in main-screen) (in final)))
Estado inicial:	(preconds (in initial))

Meta:	(effects (in final))

Tabla 10. Conjunto de transiciones para alcanzar la meta desde el estado inicial.

Planificación:
A0
A1
End.

El conjunto de pasos de esta propuesta y sus resultados se muestra en la tabla 11.

Tabla 11. Conjunto de pasos para la generación de pruebas.

Paso	**Descripción**	**Resultado**
1	Construcción del diagrama de estados a partir del caso de uso.	Diagrama de estados.
2	Descripción de los estados mediante proposiciones.	Conjunto de proposiciones del diagrama de estados.
3	Definición de proposiciones adicionales.	Conjunto de proposiciones extendido.
4	Generación de operaciones.	Conjunto de operaciones posibles sobre el conjunto de proposiciones.
5	Especificación del estado inicial y meta.	Conjunto de proposiciones que definen el estado inicial y la meta del sistema.
6	Definición del criterio de cobertura.	Criterio de cobertura.
7	Aplicación del algoritmo de generación de pruebas.	Pruebas del sistema.

La ilustración 4 describe el orden de ejecución de los pasos.

A continuación, se describen con más detalle los pasos de la tabla 11.

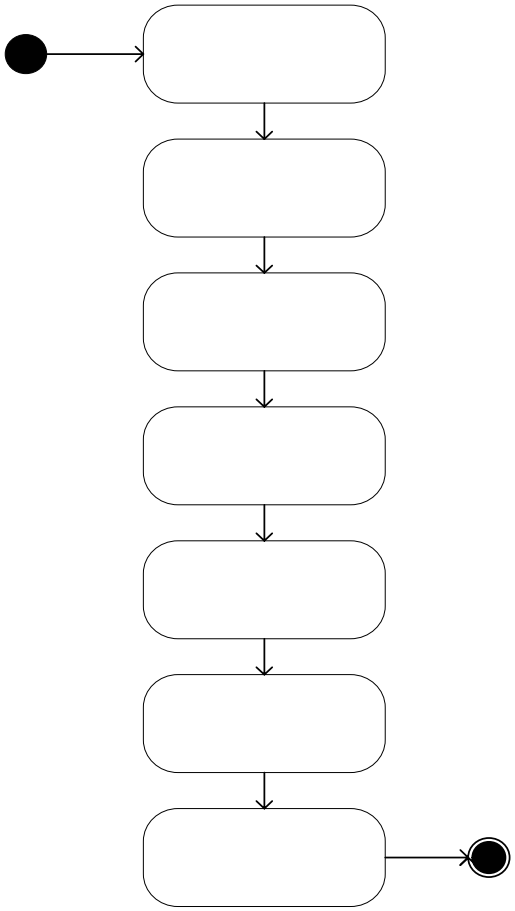
El primer bloque de esta propuesta engloba el punto primero. El segundo bloque engloba los 6 puntos restantes.

En el primer bloque, esta propuesta incluye una descripción resumida del conjunto de reglas a aplicar para generar un diagrama de estados a partir de la definición de un caso de uso. Este proceso se detalla con más extensión en [Fröhlich99].

En el segundo bloque se realiza una traducción a STRIPS [Fikes71] del diagrama de estados obtenido y se aplica un algoritmo para la generación de pruebas. Este algoritmo, descrito en la propuesta, puede ser ejecutado en cualquier herramienta de planificación que soporte STRIPS. STRIPS es un lenguaje utilizado en la

especificación de problemas de planificación. Este lenguaje se encuentra actualmente soportado por un amplio conjunto de herramientas. En los siguientes párrafos se describen con más detalle los pasos del segundo bloque.

Ilustración 4. Diagrama de actividades.



Según esta propuesta, tres tipos de diagramas UML son los más adecuados para describir el comportamiento de un caso de uso. Estos diagramas son: diagramas de estados, diagramas de acciones y diagramas de interacción. Los motivos expuestos en esta propuesta para seleccionar los diagramas de estados son que permiten visualizar varios escenarios de uso, permiten definir varias alternativas de ejecución, incluido el tratamiento de errores, de una manera más simple y completa que los diagramas de actividades, que permiten visualizar mejor las interacciones de los usuarios con el sistema. Además, la notación de máquinas de estados dentro de estados permite modelar de manera sencilla comportamiento que involucra a varios casos de uso.

La construcción del diagrama de estados a partir de la descripción en lenguaje natural del caso de uso se realiza aplicando un conjunto de reglas definidas en la propuesta.

Una vez construido, este diagrama de estados se traduce a proposiciones. Una proposición es un identificador para cada precondición y poscondición presente en el diagrama de estados. La propuesta no detalla cómo extraer las proposiciones a partir del diagrama de estados. Además, serán necesarias proposiciones adicionales para este dominio de problema como las que se muestran en la tabla 12.

Tabla 12. Proposiciones adicionales para la generación de casos de prueba.

Proposición	Descripción
Int(E)	Cierto si el estado activo es el estado E
Log_stat(E)	Cierto si alguna vez el estado activo ha sido E
Log_tran(T)	Cierto si alguna vez se ha ejecutado la transición T.

Además de estas proposiciones, pueden ser necesarias más proposiciones adicionales. Existen proposiciones que no dependen de los estados ni transiciones del diagrama de estados, sino que dependen, por ejemplo, de las propiedades de los conjuntos de datos de prueba o de las entradas del sistema. En este caso es necesario añadir dos nuevas proposiciones una en la que el conjunto de datos es válido, o está definido y otra en donde el conjunto de datos no es válido o no está definido.

Una operación se define como un conjunto de requisitos, adicciones y sustracciones. El conjunto de requisitos son las proposiciones que deben cumplirse para poder aplicar la operación. Las adicciones son el conjunto de proposiciones que van a añadirse al sistema al ejecutar la operación. Las sustracciones son el conjunto de proposiciones que desaparecen del sistema al ejecutar la operación. Todas las transiciones se modelan mediante operaciones. La traducción del diagrama de estados a un conjunto de operaciones, una vez identificadas las proposiciones a utilizar, se realiza de manera sistemática aplicando el conjunto de reglas definidas en esta propuesta. El objeto de esta traducción es poder emplear técnicas de inteligencia artificial en la generación de las pruebas; en concreto técnicas de planificación.

El estado inicial define las proposiciones ciertas al principio de la ejecución de la máquina de estados. La meta define el estado final de las proposiciones. En el proceso de generación de pruebas, un programa de inteligencia artificial intenta encontrar el

conjunto de operaciones (es decir, transiciones en el diagrama de estados) que, a partir de las proposiciones del estado inicial, permite obtener las proposiciones de la meta.

Existen varios criterios para establecer las coberturas que proporcionarán las pruebas generadas sobre el diagrama de estados. Los dos criterios principales son el criterio de considerar todos los estados y el criterio de considerar todas las transiciones.

El algoritmo de generación de pruebas presentado en la propuesta se basa en el criterio de considerar todas las transiciones. Este algoritmo consiste en encontrar un conjunto de operaciones que, aplicadas a partir del estado inicial, permitan obtener las proposiciones especificadas en la meta.

La ventaja principal de esta propuesta es que su proceso de generación de pruebas se explica de una manera sencilla y se ilustra con un ejemplo descriptivo. Sin embargo, solo permite generar pruebas a partir de un caso de uso de manera aislada, sin tener en cuenta posibles dependencias con otros casos de uso. En la tabla 13 se presentan de manera esquematizadas las ventajas e inconvenientes que pueden extraerse del estudio detallado de la propuesta.

Tabla 13. Resumen de ventajas e inconvenientes.

Ventajas.
• El proceso de generación de pruebas se explica claramente. • Este proceso se ilustra con un ejemplo descriptivo. • El método puede ser aplicado sobre cualquier herramienta de planificación con soporte para STRIPS. • Ofrece una guía para establecer la cobertura de las pruebas generadas.
Inconvenientes.
• Es una propuesta pensada para tratar los casos de uso de manera aislada. • Cuando se tiene un conjunto de casos de uso relacionados, es necesario incluir sub-máquinas de estados lo que aumenta la complejidad. • Sólo automatiza el proceso de generación de pruebas, pero la traducción del caso de uso al diagrama de estados debe hacerse de manera manual. • La traducción del diagrama de estados en operaciones también debe hacerse de manera manual. • Esta traducción no es sistemática al cien por cien, ya que no ofrece reglas claras y precisas para incluir operadores que no dependan de los diagramas, como en el caso de los datos de prueba. • Múltiples conjuntos de datos de prueba o conjuntos de datos complejos aumentan el número de proposiciones y operaciones y la complejidad del proceso. • No ofrece ninguna guía sobre como implementar las pruebas generadas. • No expresa las pruebas de una manera que sea fácilmente implementable. Al expresar las pruebas mediante proposiciones y operaciones, es necesario volver atrás para describir dichas operaciones.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.2. Requirements by Contract

Esta propuesta [Nebut03], llamada a partir de ahora RBC, se divide en dos partes. En la primera propone extender los casos de uso de UML mediante contratos. Estos contratos incluyen pre y poscondiciones y casos de uso parametrizados. En la segunda parte se muestra cómo generar automáticamente casos de prueba a partir de los casos de uso extendidos.

El trabajo [Nebur04] describe cómo aplicar esta propuesta a familias de productos.

El punto de partida de esta propuesta es un diagrama de casos de uso según la notación de UML [UML03]. Al final del proceso descrito en esta propuesta se obtiene, un modelo de casos de uso extendido con contratos y un conjunto de casos de prueba que verifican la implementación de dicho modelo de caso de uso. Los casos de uso se expresan como caminos de ejecución que recorren secuencias de casos de uso que satisfagan sus precondiciones y poscondiciones. Un ejemplo de los caminos generados por la herramienta de soporte se muestra en la tabla 14.

Tabla 14. Casos de prueba generados mediante RBC.

[RemoveBook(mary, emma), AddBook(mary, beloved)]
[Register(peter), AddBook(mary, beloved)]
[Register(peter), RemoveBook(mary, emma)]
[Register(jane), AddBook(mary, beloved)]
[Register(jane), RemoveBook(mary, emma)]
[Register(jane), Register(peter)]
[AddBook(mary, beloved), RemoveBook(mary, emma), RemoveBook(mary, beloved)]
[AddBook(mary, beloved), Register(jane), RemoveBook(mary, beloved)]
[AddBook(mary, beloved), Register(jane), RemoveBook(mary, emma)]
[AddBook(mary, beloved), Register(jane), Register(peter)]
[RemoveBook(mary, emma), Register(peter), AddBook(mary, beloved)]
[RemoveBook(mary, emma), Register(jane), AddBook(mary, beloved)]
[RemoveBook(mary, emma), Register(jane), Register(peter)]
[…]

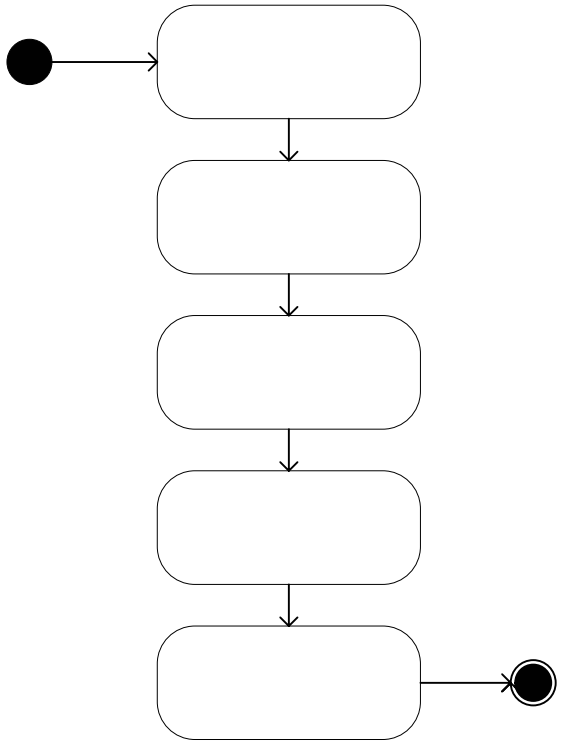
El conjunto de pasos de esta propuesta y su resultado se muestra en la tabla 15.

Tabla 15. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Extensión de los modelos de casos de uso mediante contratos.	Casos de uso con parámetros, precondiciones y poscondiciones.
2	Construcción del modelo de ejecución de casos de uso.	Modelo de ejecución de casos de uso.
3	Selección del criterio de cobertura.	Criterio de cobertura para recorrer el modelo de ejecución.
4	Aplicación del criterio al modelo de ejecución	Secuencias de casos de uso válidas.
5	Generación de pruebas	Pruebas ejecutables sobre el sistema.

La ilustración 5 describe el orden de ejecución de los pasos.

Ilustración 5. Diagrama de actividades.



Esta propuesta solo describe los cuatro primeros pasos. El quinto paso debe ser realizado con un generador de pruebas. Este quinto paso no se describe en la propuesta.

En el primer paso, los casos de uso se extienden mediante un lenguaje de contratos que permite incluir precondiciones, poscondiciones y parámetros. Estas pre y poscondiciones se expresan mediante proposiciones. Con estos elementos es posible expresar las dependencias existentes entre los casos de uso.

El modelo de ejecución de casos de uso es un diagrama que expresa el comportamiento del sistema a partir de sus casos de uso. Cada nodo es el estado del sistema, el cual viene determinado por las proposiciones y cada transición es una instancia de caso de uso, es decir, la ejecución de un caso de uso.

RBC propone cuatro criterios distintos para recorrer el modelo de ejecución y obtener casos de pruebas. Estos cuatro criterios se describen en la tabla 16.

Tabla 16. Criterios para la generación de pruebas.

Criterio	Descripción
Criterio de todos los bordes.	Existe al menos un objetivo de prueba (de donde salen los objetivos de prueba ¿) por cada transición
Criterio de todos los vértices.	Existe al menos un objetivo de prueba por cada vértice.
Criterio de todos los casos de uso instanciados.	Existe al menos un objetivo de prueba por cada caso de uso instanciado.
Criterio de todos los vértices y casos de uso instanciados.	Existe al menos un objetivo de prueba por cada caso de uso instanciado y por cada vértice.

Una vez aplicado el criterio de cobertura, se obtienen un conjunto de instancias de casos de uso con sus correspondientes parámetros. El último paso, que esta propuesta no detalla, es generar casos de prueba a partir de estas instancias mediante herramientas de generación de pruebas.

Los principales puntos fuertes de RBC son sus distintos criterios de cobertura y la existencia de una herramienta de soporte, aún muy preliminar. RBC permite generar pruebas que involucren secuencias de casos de uso. Su principal punto débil es no permitir desarrollar pruebas para verificar el comportamiento de cada caso de uso de manera aislada. En la tabla 17 se presentan de manera esquematizadas las ventajas e inconvenientes que pueden extraerse del estudio detallado de la propuesta.

Tabla 17. Resumen de ventajas e inconvenientes.

Ventajas.
• Ofrece una herramienta experimental de libre descarga con la implementación de los criterios propuestos. • Permite generar pruebas basadas en secuencias de casos de uso. • Los contratos son un medio muy flexible para expresar las dependencias de un caso de uso respecto de otros casos de uso.
Inconvenientes.
• Extiende los casos de uso de una manera no estándar según los mecanismos de extensión de UML.

- No describe como generar pruebas que verifiquen la implementación de un caso de uso de manera aislada.
- No ofrece ningún criterio que indique cuantos parámetros distintos debe utilizar un caso de uso.
- No existe ninguna relación entre los parámetros simbólicos utilizados y sus valores reales.
- Utiliza muchos conceptos, objetivo de prueba, casos de uso instanciados, definidos de manera muy pobre, o no definidos.
- No detalla como implementar las pruebas.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.3. Testing From Use Cases Using Path Analysis Technique

El punto de partida de esta propuesta [Naresh02] es un caso de uso descrito en lenguaje natural. Esta propuesta no indica ninguna plantilla ni reglas para describir el caso de uso, pero sí indica la información más común a incluir a la hora de redactar casos de uso.

Al final del proceso de generación de pruebas se obtiene una tabla con todos los caminos posibles que deben ser probados para un caso de uso. Estos caminos también se describen en lenguaje natural sin adoptar ninguna representación formal.

Los pasos de esta propuesta se resumen en la tabla 18. La ilustración 6 describe el orden de ejecución de los pasos.

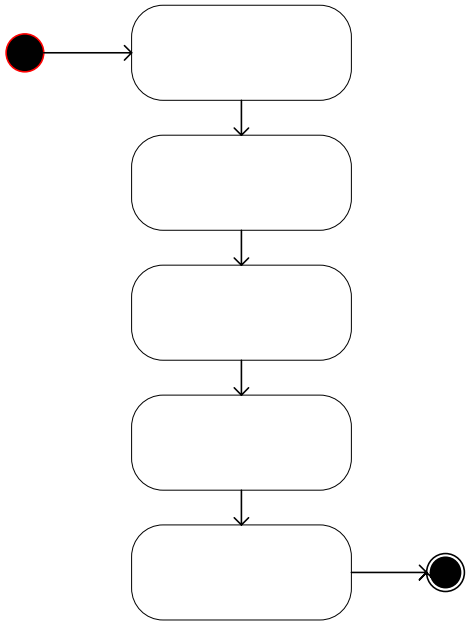
Tabla 18. Conjunto de pasos para la generación de pruebas.

Pasos	Descripción	Resultado
1	Elaboración de un diagrama de flujo a partir del caso de uso.	Diagrama de flujo.
2	Identificación de todos los posibles caminos de ejecución.	Listado de todos los caminos posibles para recorrer el diagrama de flujo.
3	Análisis y puntuación de los caminos.	Listado de caminos con su puntuación.
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.

A continuación, en los siguientes párrafos, se describe con más detalle los pasos de la tabla 18.

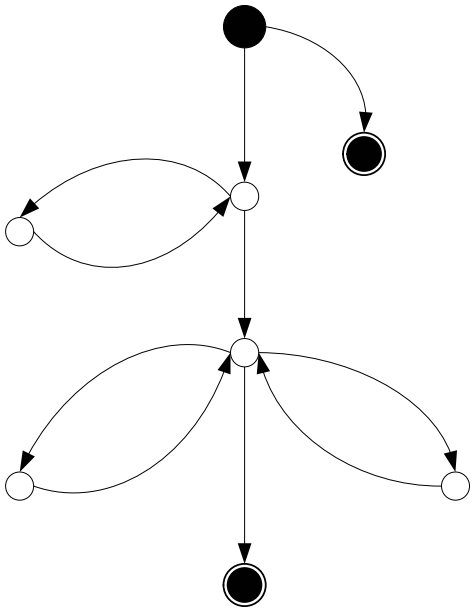
Un caso de uso tiene varios caminos de ejecución y cada uno es un caso de prueba potencial. Sin embargo, un caso de uso expresado mediante una plantilla de texto es difícil de analizar. Para facilitar el análisis, en el primer punto se elabora un diagrama de flujo de ejecución.

Ilustración 6. Diagrama de actividades.



El diagrama de flujo de ejecución sigue una notación propia descrita en la propuesta. En dicho diagrama de flujo se representan como nodos los puntos donde el flujo de ejecución se bifurca, o varios flujos de ejecución convergen. Se representa como rama la acción a desarrollar dentro de dicho caso de uso. Un ejemplo de diagrama de flujo se muestra en la ilustración 7.

Ilustración 7. Ejemplo de diagrama de flujo.



A partir de este diagrama de flujo de ejecución, mediante el análisis de caminos se identifican todos los caminos diferentes que lo recorre y se construye una lista.

Después se analizan y puntúan los caminos. Esta propuesta propone asignar una serie de atributos a cada uno de los caminos, puntuar dichos atributos y obtener la puntuación final de cada camino. Esta propuesta deja abierta la posibilidad de analizar tantos atributos cómo se considere conveniente, cómo puntuarlos y cómo obtener la puntuación final.

Mediante esta puntuación es posible ordenar los caminos según su importancia y descartar caminos poco importantes para reducir el número de pruebas. En la tabla 19 se muestra un ejemplo de caminos y sus puntuaciones, obtenidos del diagrama de flujo de la ilustración 7. Los dos atributos puntuados, importancia y frecuencia, son los descritos en la propuesta y la escala de puntuación está comprendida entre el 1 y el 10.

Tabla 19. Ejemplo de caminos y puntuaciones.

Id	Importancia	Frecuencia	Nombre	Descripción
1	10	1	2	Fallo de página
2	10	6	1, 4, 7	Acceso correcto al sistema

Por último, se implementan los caminos seleccionados como casos de prueba. Será necesario añadir valores de prueba, por lo que puede existir más de un caso de prueba aplicando distintos valores de prueba a un mismo camino. La propuesta no entra

en detalle sobre como realizar este último paso ni indica cual sería el resultado obtenido o como estaría descrito.

Las principales ventajas de esta propuesta son que ofrece un método sencillo, y permite seleccionar las pruebas en función de la importancia de los caminos. Sin embargo es una propuesta muy sencilla y es difícil de escalar a sistemas de gran tamaño o complejidad. En la tabla 20 se presentan de manera esquematizadas las ventajas e inconvenientes que pueden extraerse del estudio detallado de la propuesta.

Tabla 20 Resumen de ventajas e inconvenientes.

Ventajas.
• Es un proceso sencillo, rápido de explicar y fácil de poner en marcha. • Ofrece una guía para descartar caminos que aportan poco al proceso de prueba. • Incluye un ejemplo de aplicación paso a paso. • Permite probar de forma exhaustiva todo el comportamiento asociado a un caso de uso.
Inconvenientes.
• No se incluyen referencias a proyectos reales donde se haya aplicado esta propuesta • No indica como implementar las pruebas. • Todo el proceso, salvo el diagrama de flujo de ejecución, se realiza mediante lenguaje natural, lo que da pie a ambigüedades y es un handicap a la hora de automatizar el proceso con herramientas. • No indica como desarrollar pruebas con casos de uso que dependan de otros casos de uso. • No ofrece una guía sobre como seleccionar y cuantificar los atributos ni cómo obtener la puntuación final. • No indica como generar caminos en diagramas de flujo de ejecución que presenten bucles. • No indica como generar los valores de prueba ni como afectan al proceso de generación de pruebas. • No ofrece ninguna guía sobre como implementar las pruebas generadas. • No expresa las pruebas de una manera en que sean fácil de implementar. Al expresar las pruebas mediante lenguaje natural, el ingeniero de pruebas debe tener un conocimiento profundo de lo que está probando para poder interpretarlas adecuadamente.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.4. Test Cases From Use Cases.

Esta propuesta [Heumann02] muestra cómo utilizar los casos de uso para generar casos de prueba y cómo este proceso puede comenzar en etapas tempranas del ciclo de desarrollo.

El punto de partida de esta propuesta es un caso de uso definido textualmente mediante una plantilla y lenguaje natural. Esta propuesta no impone un modelo formal de representación de un caso de uso, pero sí indica los elementos que deben aparecer en un caso de uso.

Al final del proceso descrito en esta propuesta se obtiene una tabla donde se describe, lenguaje natural, los casos de prueba para verificar la correcta implantación del caso de uso.

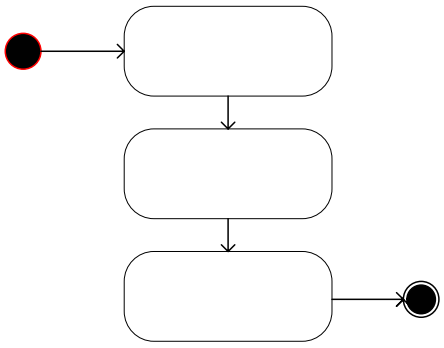
El conjunto de pasos de esta propuesta se describe en la tabla 21. La ilustración 8 muestra el orden de ejecución de los pasos.

Tabla 21. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Generación de escenarios de uso.	Un conjunto de escenarios de prueba por cada caso de uso.
2	Identificación de casos de prueba.	Un conjunto de casos de prueba por cada uno de los escenarios del paso anterior y las condiciones necesarias para su ejecución.
3	Identificación de valores de prueba.	Conjunto de valores de prueba necesarios para cada uno de los casos de prueba anteriores.

Esta propuesta define escenario de uso como una instancia particular de un caso de uso. Un caso de prueba se define como un conjunto de entradas, condiciones de ejecución y resultados esperados.

Ilustración 8. Diagrama de actividades.



En el primer paso se identifican todas las posibles combinaciones entre el camino principal de ejecución y los caminos alternativos descritos en un caso de uso. Estas combinaciones se expresan mediante una tabla.

La propuesta obliga a que exista, al menos, un caso de prueba por cada escenario identificado en el paso anterior. Se deben identificar las condiciones que deben

cumplirse para qué se ejecute el escenario de uso, por ejemplo datos válidos o inválidos u otro tipo de condiciones. Toda esta información se expresa mediante tablas sin ninguna notación o formalismo.

Por último, la las tablas anteriores se les añade los valores que se utilizarán en las pruebas. La propuesta no se indica como obtener estos datos.

La principal ventaja de esta propuesta es su sencillez y su principal inconveniente la falta de detalle y rigor en la descripción del proceso y la falta de escalabilidad. En la tabla 22 se presentan de manera esquematizadas las ventajas e inconvenientes que pueden extraerse del estudio detallado de la propuesta.

Tabla 22. Resumen de ventajas e inconvenientes.

Ventajas.
• Esta propuesta es sencilla de entender y rápida de aplicar. • Al buscar todas las combinaciones posibles ofrece una cobertura total del caso de uso.
Inconvenientes.
• Es una propuesta pensada para tratar los casos de uso de manera aislada. No es capaz de reflejar dependencias entre casos de uso • Al estar basado por completo en lenguaje natural es difícil de automatizar. • Para casos de uso complejos, es difícil identificar todas las posibles combinaciones. • Para casos de uso complejos el número de casos de prueba es muy elevado. • Aunque menciona que este proceso puede ser aplicado en etapas tempranas del desarrollo, no indica cómo hacerlo. • En ningún momento ofrece una guía o reglas sistemáticas para realizar los pasos.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.5. PLUTO y Category Partition Method

La propuesta Product Lines Use case Test Optimization (PLUTO) [Bertolino03] y [Bertolino04] describe como generar casos de pruebas para familias de productos.

El punto de partida son casos de uso para familias de productos. Esta propuesta extiende la notación de casos de uso con plantillas y lenguaje natural recogida en [Cockburn00] para expresar los elementos comunes a todos los productos y los puntos de variabilidad donde cada producto presenta unas características concretas distintas de

los demás productos. Los casos de uso para familias de productos se describen con más detalle en [Bertolino02].

El resultado de esta propuesta es un conjunto de casos de prueba comunes para todos los productos de la familia y, además, un conjunto de casos de prueba específicos para cada producto de la familia. Estos casos de prueba se describen mediante plantiíllas en lenguaje natural

Las pruebas generadas por esta propuesta son descripciones abstractas de un caso de prueba que deberá ser refinado para obtener un caso de prueba ejecutable sobre el sistema. Sin embargo PLUTO no indica ni ofrece referencias de cómo realizar este proceso.

El proceso de generación de casos de prueba propuesto en PLUTO es una adaptación del proceso de partición de categorías (Category Partition o CP) [Ostrand 88] puesto al día para trabajar con especificaciones modeladas mediante casos de uso para familias de productos.

Los 8 pasos de esta propuesta son básicamente los mismos que los pasos de CP y se describen en la tabla 23. La ilustración 9 describe el orden de ejecución de los pasos.

Tabla 23. Conjunto de pasos para la generación de pruebas.

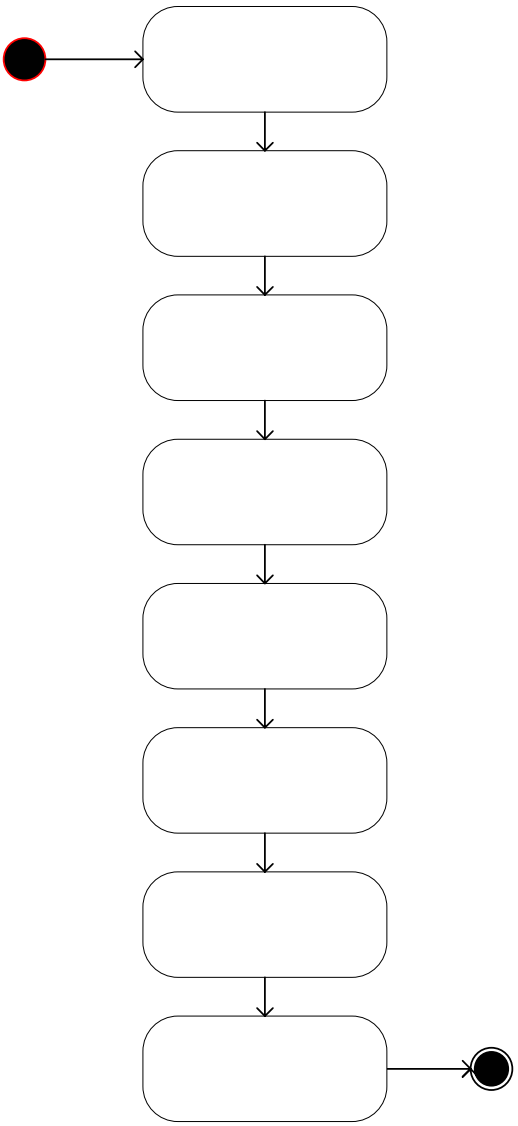
Paso	Descripción	Resultado
1	Descomposición de la especificación funcional del sistema en unidades funcionales	Unidades funcionales que pueden probarse de manera independiente.
2	Identificación de parámetros y condiciones del entorno que afectan a la ejecución de cada unidad funcional.	Conjunto de entradas del sistema y estado del sistema necesario para la ejecución de cada unidad funcional.
3	Búsqueda de categorías.	Una categoría para cada parámetro y condición del sistema que indicará el rango que de valores que podrá tomar.
4	División de cada categoría en elecciones.	Conjunto de elecciones de cada categoría.
5	Identificación de restricciones.	Lista de restricciones de cada elección.
6	Redacción de especificaciones de prueba.	Listado de especificaciones de prueba.
7	Generación de marcos de prueba.	Marcos de prueba a partir de las especificaciones.
8	Generación de scripts de prueba.	Conjunto de scripts, cada uno de ellos conteniendo varias de las pruebas generadas.

Los pasos 5, 6 y 7 pueden repetirse varias veces para refinar las especificaciones de prueba.

El primer paso no se aplica en PLUTO, ya que esta propuesta considera que las unidades funcionales corresponden con los casos de uso.

En el segundo, tercer y cuarto paso, la identificación de categorías y elecciones para cada categoría deben realizarla los ingenieros de pruebas a partir de su experiencia y conocimiento del sistema a probar. Cada elección es un subconjunto de los datos de cada categoría

Ilustración 9. Diagrama de estados.



Las restricciones identificadas en el paso quinto expresan dependencias entre las elecciones. Estas restricciones se expresan mediante cláusulas IF booleanas asociadas a cada elección. También se utilizan restricciones para clasificar determinados tipos de condiciones. Una de las restricciones más utilizadas es “error” la cual identifica a

condiciones erróneas. Las elecciones con una restricción errónea no se aplican en todas las combinaciones posibles para reducir el número de casos de prueba.

En el paso sexto se redactan las especificaciones de prueba. Una especificación de prueba es un documento compuesto por una plantilla definida en CP que recoge todas las categorías, las elecciones de cada una de las categorías y sus restricciones.

Los pasos séptimo y octavo no son contemplados por PLUTO, por lo que da a suponer que pueden aplicarse de la misma manera en como están descritos en CP. El séptimo paso consiste en la generación automática de marcos de prueba a partir de las especificaciones de prueba. Un marco de prueba es una posible combinación de elecciones de todas las categorías presentes en una especificación de prueba. Estos marcos de prueba se traducen, en el octavo paso, a un lenguaje ejecutable y se juntan de manera aleatoria para construir scripts de prueba que serán ejecutados sobre el sistema.

A pesar de que PLUTO ha sido diseñado para familias de productos, consideramos que es un buen ejemplo de cómo aplicar una propuesta de mediados de los años 80 a sistemas actuales. Su puesta al día de la propuesta CP puede aplicarse también a casos de uso que no pertenezcan a familias de productos.

Su principal ventaja es ofrecer un método que permite generar conjuntos de valores de prueba. Sin embargo muchos pasos se definen con ambigüedad en función de los criterios personales de los ingenieros de pruebas. En la tabla 24 se presentan de manera resumida las principales ventajas e inconvenientes de esta propuesta.

Tabla 24. Resumen de ventajas e inconvenientes.

Ventajas.
• Ofrece un ejemplo práctico no trivial. • Permite verificar el comportamiento de un caso de uso teniendo en cuenta sus dependencias con otros casos de uso. • Se puede aplicar en etapas tempranas, como la elicitación de requisitos. • Indica como reducir el número de casos de prueba generados (mediante restricciones, por ejemplo, erróneas).
Inconvenientes.
• Las elecciones están muy basadas en la experiencia de los ingenieros de pruebas y el conocimiento que estos tienen sobre el sistema. • No ofrece información sobre la cobertura de las pruebas. Al dejar la elección de valores al criterio de los ingenieros de pruebas puede no ofrecer una cobertura adecuada. • No hay referencias a herramientas de soporte existentes • Aunque puede ser automatizada, al dejar muchos elementos a la decisión de los ingenieros de

pruebas no puede alcanzar la automatización total.

- Si los casos de uso no manipulan datos, esta propuesta se muestra inútil.
- Puesto que las elecciones dependen de los criterios de los ingenieros de pruebas no se puede estimar a priori el número de casos de prueba necesarios para verificar la implementación de un caso de uso.
- Falta una guía sistemática que indique como identificar categorías y elecciones.
- La inclusión de clasificaciones erróneas en un número mayor o menor de casos de prueba depende del criterio del ingeniero de prueba.
- No explica como mezclar y agrupar las pruebas en scripts.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.6. SCENario-Based Validation and Test of Software (SCENT)

SCENT [Glinz99] y [Ryser03] es un método basado en escenarios para elicitar, validar y verificar requisitos y desarrollar casos de pruebas a partir de los mismos. SCENT puede dividirse en dos partes, la parte de creación y refinado de escenarios y la parte de generación de pruebas.

Dado que SCENT incluye un proceso para la creación y refinado de escenarios, el punto de partida es un conjunto de necesidades por parte de un conjunto de clientes o usuarios. El resultado de esta propuesta es un conjunto de escenarios refinados y un conjunto de casos de prueba para verificar la correcta implementación de dichos escenarios. Estas pruebas se describen mediante tablas redactadas en lenguaje natural.

Las definiciones de escenario y de caso de uso propuestas por este método se recogen en la tabla 25.

Tabla 25. Definiciones de SCENT.

Escenario: Conjunto ordenado de interacciones entre partes, generalmente entre un sistema y un conjunto de actores externos al sistema. Puede expresar una secuencia concreta de interacciones (instancia de escenario) o un conjunto de posibles interacciones (escenario tipo).

Caso de uso: Secuencia de interacciones entre un actor o actores y un sistema, iniciada por un actor concreto, el cual produce un resultado apreciable por un actor. Este

concepto coincide con la definición de escenario tipo.

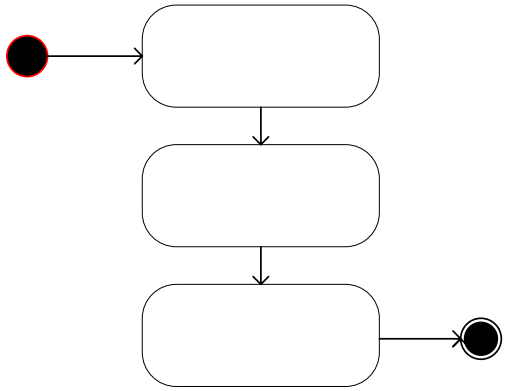
SCENT propone tres pasos para la generación de casos de prueba. Estos pasos se describen en la tabla 26.

Tabla 26. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Creación de escenarios.	Conjunto de escenarios del sistema.
2	Formalización de los escenarios.	Diagramas de estados.
3	Generación de pruebas.	Conjunto de pruebas.

La ilustración 10 describe el orden de ejecución de los pasos.

Ilustración 10. Diagrama de actividades.



Para la creación de escenarios SCENT propone un proceso de 15 tareas para la elicitación, creación y organización de los requisitos. Estas tareas y sus resultados se describen en la tabla 27.

Tabla 27. Tareas para la creación de escenarios.

Tarea	Descripción	Resultado
1	Búsqueda todos los actores que interactúan con el sistema.	Lista de actores.
2	Búsqueda de todos los eventos externos al sistema	Lista de eventos.
3	Definición de las salidas del sistema	Salidas del sistema.
4	Definición del ámbito del sistema	Ámbito del sistema.
5	Creación de resúmenes de los escenarios (tanto escenarios tipo como instancias de escenarios)	Lista de escenarios.
6	Asignación de prioridades a los escenarios.	Lista de escenarios con prioridades.

7	Transformación de instancias de escenarios en escenarios tipo y creación de una descripción paso a paso de los eventos de cada escenario	Flujos de acciones en cada escenario.
8	Creación de un diagrama resumen del sistema	Diagrama resumen del sistema
9	Revisión por parte de los futuros usuarios	Comentarios y notas sobre los escenarios.
10	Extensión de los escenarios refinando su descripción	Descripción del flujo normal de ejecución de cada escenario.
11	Modelado de flujos alternativos de ejecución	Flujos alternativos de ejecución para escenario.
12	Extracción de escenarios abstractos.	Lista de escenarios abstractos.
13	Refinado de escenarios con requisitos no funcionales, de rendimiento y de calidad.	Lista de escenarios refinados.
14	Revisión del diagrama resumen del sistema	Diagrama resumen del sistema revisado.
15	Revisión formal de los escenarios los futuros usuarios.	Escenarios validados.

Al final de este primer bloque se obtiene un conjunto de escenarios refinados y validados. Estos escenarios se expresan mediante plantillas en lenguaje natural. La estructura de estas plantillas se incluye dentro de la documentación de SCENT.

En el segundo paso, se construye un diagrama de estados por cada escenario obtenido en el primer paso. Este proceso es definido en esta propuesta como un proceso informal y creativo que no puede formalizarse, por lo que SCENT no ofrece ninguna guía precisa para su realización. Por este mismo motivo es difícil su automatización.

La construcción de los diagramas de escenarios puede desarrollarse paralelamente al paso anterior cuando comiencen a estar disponibles los primeros escenarios. A continuación estos diagramas de estados son complementados con anotaciones sobre precondiciones, entradas y salidas de datos y sus rangos válidos y requisitos no funcionales. SCENT define una extensión a la notación UML de diagrama de estados. Esta extensión no sigue el estándar para extensiones de UML.

El tercer paso está compuesto por tres tareas descritas en la tabla 28. Las dos primeras tareas son obligatorias y la restante opcional.

Tabla 28. Tareas para la generación de pruebas.

Tarea	Descripción
1	Generación de casos de prueba a partir de los diagramas de estados.
2	Generación de prueba de dependencias entre escenarios y pruebas adicionales (basadas en

	requisitos no funcionales).
3	Integración de diagramas de estados y generación de casos de prueba a partir de los diagramas de estados resultantes.

En la primera tarea los casos de prueba se generan a partir de los caminos transversales que recorren el diagrama de estados. De esta manera se recorren todos los nodos y enlaces del diagrama de estados.

El resultado de esta tarea es una tabla en lenguaje natural donde cada línea es un caso de prueba a implementar. Las columnas de la tabla serán las precondiciones de la prueba, las entradas y acciones del sistema y la salida esperada. Según SCENT, mediante este método sólo es posible generar secuencias válidas del sistema, por lo que sería necesario incluir manualmente pruebas que incluyan secuencias no válidas.

En la segunda tarea, se generan pruebas a partir del diagrama de dependencias. La estrategia para generar pruebas es similar al caso anterior, y generar pruebas mediante recorridos transversales. A pesar de que este paso se clasifica como obligatorio, SCENT no ofrece ningún ejemplo ni caso práctico de como hacerlo ni puesta como se definen las pruebas así generadas.

Respecto a la tercera tarea, tampoco se ofrece ninguna información.

Una aplicación de esta propuesta en dos proyectos reales se documenta en [Itschner98].

La principal ventaja de esta propuesta es ofrecer un método extenso y completo para la elicitación y descripción de escenarios. Sin embargo la generación de casos de prueba está pobremente descrita. En la tabla 29 se presentan de manera resumida las principales ventajas e inconvenientes de esta propuesta.

Tabla 29. Resumen de ventajas e inconvenientes.

Ventajas.
• Expone un método claro y detallado para la creación, definición y validación de los requisitos funcionales. • Incluye ejemplos prácticos y aplicaciones en casos reales. • Permite verificar el comportamiento de un caso de uso teniendo en cuenta sus dependencias con otros casos de uso. • Describe cómo aplicarla en etapas tempranas, como la elicitación de requisitos. • Esta propuesta detalla los problemas que suelen aparecer en la industria del desarrollo de software y se adapta a intentar solucionarlos.

Inconvenientes.
• Aunque la parte dedicada a los escenarios es rica y completa, la parte dedicada a la generación de pruebas es muy pobre. • SCENT define mucho de los pasos como informales y no ofrece guías claras y precisas sobre como realizarlos. Esto afecta negativamente a la automatizabilidad. • No garantiza la cobertura total de las pruebas. Podemos suponer que, al recorrer al menos una vez todos los nodos y enlaces se está cubriendo toda la posible funcionalidad, pero como el propio método expone, esto no es así. • No hay referencias a herramientas de soporte. • Aunque puede ser automatizada, al dejar muchos elementos a la decisión de los ingenieros de pruebas no puede alcanzar la automatización total. • No indica como reducir el número de casos de prueba generados. • Los distintos documentos son inconsistentes entre sí. Los pasos del proceso de generación de pruebas varían de uno a otro.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.7. TOTEM.

El objetivo de esta propuesta es generar casos de prueba a partir del modelo del sistema expresado mediante diagramas con la notación UML y obtenidos en la fase de análisis del sistema.

El punto de partida de esta propuesta son artefactos desarrollados durante la fase de análisis. En concreto los artefactos que utiliza TOTEM son:

- Diagramas UML de casos de uso.
- Descripciones en lenguaje natural de los casos de uso.
- Diagramas de secuencia o colaboración por cada caso de uso.
- Diagrama de clases con las clases del dominio de la aplicación.
- Diccionario de datos con la descripción de cada clase, método y atributo.
- Restricciones expresadas en OCL.

A partir de estos artefactos, TOTEM permite obtener requisitos de prueba del sistema, casos de prueba, oracles de pruebas y constructores de pruebas.

TOTEM propone un proceso de 8 pasos para la generación de pruebas. TOTEM sólo describe 4 de ellos. Estos pasos se recogen en la tabla 30. La ilustración 11 describe el orden de ejecución de los pasos.

Tabla 30. Conjunto de pasos para la generación de pruebas.

Pasos	Descripción
1	Verificación de los modelos de análisis.
2	Derivación de secuencias de dependencias entre casos de uso.
3	Derivación de requisitos de prueba a partir de diagramas de secuencias.
4	Derivación de requisitos de pruebas a partir del diagrama de clases del sistema.
5	Definición de secuencias de variantes.
6	Derivación de requisitos para las pruebas del sistema.
7	Derivación de casos de prueba para las pruebas del sistema.
8	Derivación de oracles y hamess de prueba.

TOTEM describe sólo los pasos 2, 3 y 5. Cada uno de estos pasos se divide en varias tareas. Al final de cada uno de los pasos se obtiene un conjunto de artefactos necesarios para la generación de pruebas del sistema.

El primer paso descrito por TOTEM es el paso 2 de la tabla 30. En este paso se identifican dependencias secuenciales de casos de uso, esto es, que casos de uso dependen de otros para poder realizarse. Este paso se subdivide en las tareas recogidas en la tabla 31.

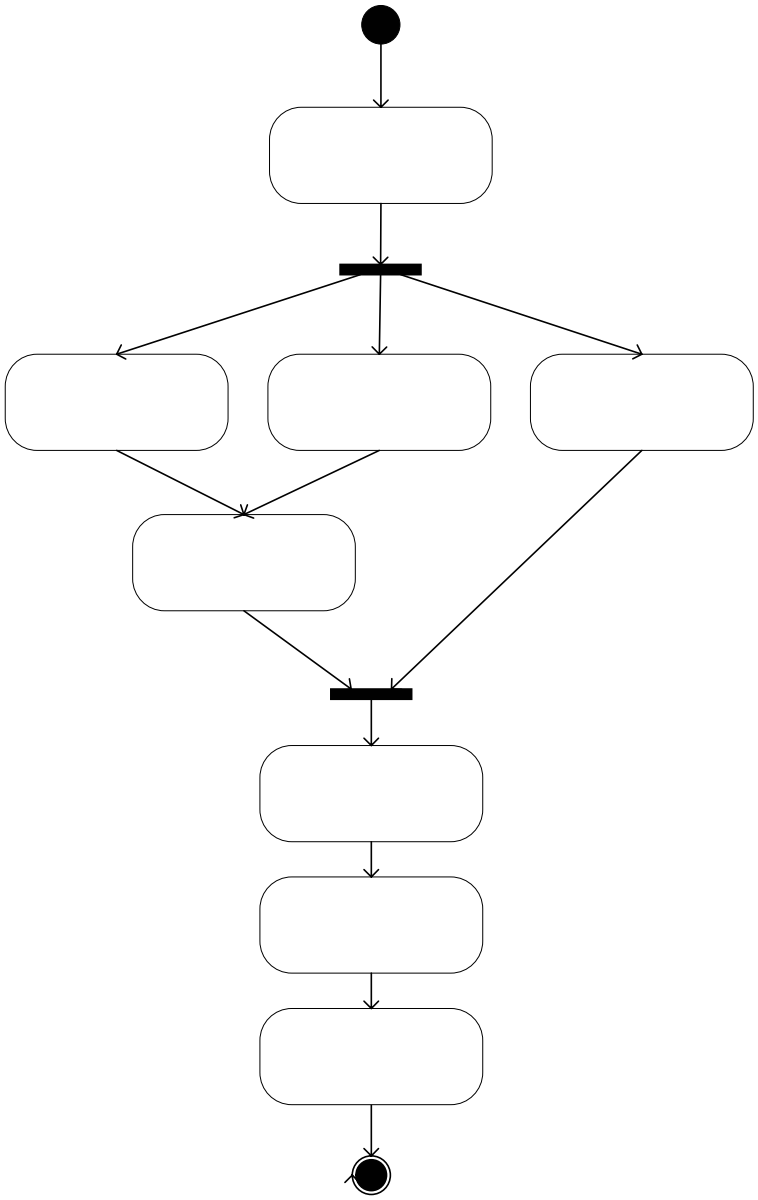
Tabla 31. Tareas para la derivación de secuencias de dependencias.

Tarea	Descripción
1	Representación de dependencias secuenciales de los casos de uso.
2	Generación de secuencias de casos de uso.

Las dependencias entre casos de uso se expresan mediante un diagrama de actividades donde cada actividad representa un caso de uso, y parámetros asociados a cada caso de uso. TOTEM propone construir un diagrama de actividades por cada actor del sistema.

Una vez identificadas las dependencias entre casos de uso, mediante diagramas de actividades y casos de uso parametrizados, se procede a generar secuencias válidas de casos de uso. Las secuencias se obtienen analizando todos los posibles caminos del diagrama de actividades de la tarea anterior mediante recorridos en profundidad.

Ilustración 11. Diagrama de actividades.



La búsqueda de secuencias se repite más de una vez aplicando varios recorridos en profundidad sobre el diagrama de actividades. En el caso de que las secuencias incluyan parámetros, estos parámetros serán distintos en cada secuencia. El número de repeticiones y el número de parámetros distintos se deja al criterio de los ingenieros de prueba.

Una vez obtenidas las secuencias, se combinan para generar secuencias mayores. Las combinaciones deben preservar las dependencias de los casos de uso y evitar la duplicación de instancias de los casos de uso, es decir, que aparezca más de una vez el mismo caso de uso con el mismo conjunto de parámetros.

Deriva
secuen
dependen
casos

En el tercer paso de la tabla 30, a partir de los diagramas de colaboración o de secuencia asociados a cada caso de uso se generan secuencias de escenarios, también llamadas secuencias de instancias de casos de uso.

TOTEM muestra cómo realizar este paso a partir de un diagrama de secuencia UML que recoge todas las posibles alternativas descritas en el caso de uso. Este paso se divide en cinco tareas recogidas en la tabla 32.

Tabla 32. Tareas para la derivación de secuencias de escenarios.

Tarea	Descripción
1	Expresión de los diagramas de secuencia mediante expresiones regulares.
2	Identificación de las condiciones de realización de los caminos.
3	Especificación de secuencias de operación.
4	Identificación de oracles de prueba.
5	Construcción de tablas de decisión.

En la primera tarea de la tabla 32 se traduce la información contenida en el diagrama de secuencias a expresiones regulares, obteniendo una expresión regular por cada diagrama. Cada una de las posibles secuencias de eventos del diagrama se expresa como una suma de productos dentro de la expresión regular.

A continuación, en la segunda tarea, se obtiene una expresión regular compuesta de varios términos. Cada uno de esos términos es una variante en la secuencia de interacciones del diagrama de secuencia. Cada uno de los términos tiene asociadas un conjunto de condiciones que indicará si se ejecuta dicha alternativa o no. En esta tarea se identifican dichas condiciones y se expresan mediante expresiones OCL.

En la tercera tarea, a partir de las condiciones para la ejecución de cada una de las interacciones, se especifican las secuencias de operaciones concretas que serán ejecutadas para cada término. Esto se consigue sustituyendo operadores de expresiones regulares, como * o +, por secuencias precisas. Cada una de las veces que se repiten estas secuencias (las de * o +) son una secuencia distinta de un mismo término.

El concepto de oracle de prueba (test oracle) [Binder00] no está definido en TOTEM, por lo que asumimos que es el mismo concepto que se maneja habitualmente en entornos de pruebas. TOTEM propone generar oracles de prueba por cada secuencia obtenida en la tarea anterior. Estos oracles se generan a partir de las poscondiciones de una secuencia de interacciones.

En la quinta tarea se formaliza toda la información obtenida en tablas de decisiones. Estas tablas de decisiones permiten expresar requisitos de pruebas formales y formarán parte del plan de prueba. Una tabla de decisión recoge las condiciones iniciales de ejecución que controlan las distintas alternativas y las acciones de las ejecuciones de cada una de las alternativas o variantes. Cada fila de la tabla de decisiones será una variante, o instancia del caso de uso. Los casos de prueba deberán cubrir cada una de las variantes, al menos una vez.

El último paso de la tabla 30 incluida en esta propuesta es el paso 5. Además de las tablas de decisiones son necesarias secuencias de operaciones que puedan ser ejecutadas sobre secuencias de casos de uso, esto es, que involucre a más de un caso de uso. La propuesta no describe de forma clara como generar estas propuestas, sino que se remite a repetir los pasos anteriores sobre más de un caso de uso simultáneamente.

La ventaja principal de esta propuesta es la profundidad con la que describe cada uno de los pasos, sin embargo, su principal inconveniente es que no describe el proceso completo de generación de pruebas del sistema. En la tabla 33 se presentan de manera resumida las principales ventajas e inconvenientes de esta propuesta.

Tabla 33. Resumen de ventajas e inconvenientes.

Ventajas.
• El ejemplo incluido es no trivial y detallado. • Cada uno de los pasos descritos se detalla con mucha precisión y profundidad • Es una de las pocas propuestas analizadas que genera pruebas tanto a partir la descripción de cada caso de uso como a partir de secuencias de casos de uso.
Inconvenientes.
• No es una propuesta completa ni describe todos los pasos. • Deja muchas decisiones al criterio de los ingenieros de pruebas. • La generación de pruebas a partir de diagramas de secuencias pone de relieve detalles internos del sistema que no deberían tenerse en cuenta en pruebas del sistema. • No indica criterios para reducir el número de pruebas generado. • No hay referencias a herramientas de soporte existentes • Aunque puede ser automatizada, al dejar muchos elementos a la decisión de los ingenieros de pruebas no puede alcanzar la automatización total.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.8. Requirement Base Testing.

Requirement Base Testing (RBT) [Mogyorodi01], [Mogyorodi02] y [Mogyorodi03] se divide en dos actividades principales: revisiones de requisitos y generación y revisión de casos de prueba.

El punto de partida de esta propuesta consiste en un conjunto de requisitos descritos en lenguaje natural. Esta propuesta no incluye ninguna plantilla ni propone ningún formalismo para la redacción de estos requisitos.

El resultado de esta propuesta es un conjunto de casos de prueba expresados en lenguaje natural, compuestos de una serie de causas, o estado del sistema y sus efectos, o resultados esperados.

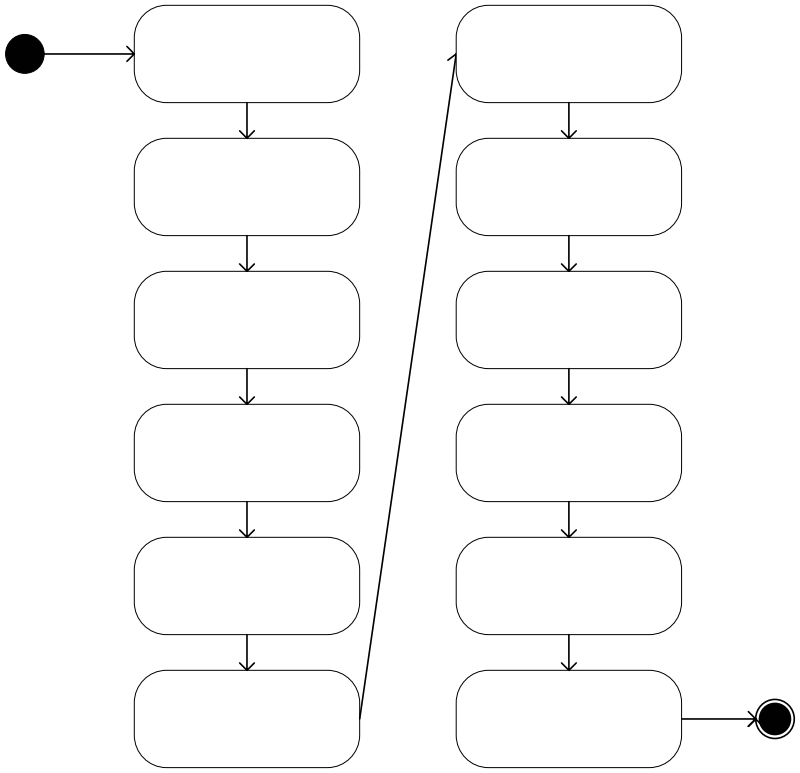
RBT consta de 12 pasos los cuales se describen en la tabla 34. En la documentación encontrada no se indica los resultados o productos que se van obteniendo en la realización de cada uno de los pasos.

Tabla 34. Conjunto de pasos para la generación de pruebas.

Tarea	Descripción
1	Validación de los requisitos respecto de los objetivos.
2	Aplicación de casos de uso sobre los requisitos.
3	Revisión inicial de ambigüedad.
4	Revisión por parte de los expertos de dominio.
5	Creación de diagramas de causa-efecto y generación de pruebas.
6	Verificación de consistencia.
7	Revisión de los casos de prueba por los escritores de los requisitos.
8	Revisión de los casos de prueba por los usuario del sistema.
9	Revisión de los casos de prueba por los desarrolladores.
10	Revisión de los casos de prueba sobre el modelo de diseño del sistema.
11	Revisión de los casos de prueba sobre el código.
12	Ejecución de los casos de prueba.

La ilustración 12 describe el orden de ejecución de los pasos.

Ilustración 12. Diagrama de actividades.



De todos los pasos propuestos en RBT, los que verdaderamente están relacionados con la generación de los casos de prueba serían los pasos 5 y 6. Los pasos del 1 al 4 están más relacionados con la validación de requisitos que con el proceso de generación de pruebas. Los pasos del 7 al 11 son simplemente revisiones de las pruebas generadas.

La documentación encontrada sobre esta propuesta sólo describe, a grandes rasgos, cómo realizar la revisión de ambigüedad sobre los requisitos y la creación de diagramas de causa-efecto, pasos 3 y 5, por lo que sólo comentaremos estos pasos.

La revisión de ambigüedad [Mogyorodi02] es una técnica para identificar y eliminar palabras y frases ambiguas de la descripción de los requisitos. Como se ha comentado al principio de este punto, no es necesario que el requisito cumpla ninguna condición salvo que esté expresado en lenguaje natural. Para realizar una revisión de ambigüedad se identifican y corrigen, en primer lugar, las frases y palabras potencialmente ambiguas. Después se buscan más elementos ambiguos en los requisitos y se corrigen.

Los diagramas de causa y efecto que usa esta propuesta siguen el modelo propuesto por IBM en 1973. La propuesta no indica como traducir los requisitos a modelos de causa y efecto. Una vez obtenidos estos diagramas, son traducidos a tablas

de decisiones donde se incluyen todas las causas, sus efectos y todas las posibles combinaciones de ambos. Cada una de esas combinaciones será un caso de prueba potencial.

A partir de estos casos, una herramienta comercial, sin versión de evaluación disponible, calcula el conjunto mínimo de pruebas y las genera. Tampoco existen referencias sobre que técnicas emplea dicha herramienta, ni de los resultados generados.

La ventaja principal de esta propuesta es la existencia de una herramienta de soporte madura y comercial, sin embargo la falta casi total de documentación de calidad impide su aplicación si no es mediante la adquisición de esta herramienta. En la tabla 45 se presentan de manera resumida las principales ventajas e inconvenientes de esta propuesta.

Tabla 35. Resumen de ventajas e inconvenientes.

Ventajas.
• Cuenta con una herramienta de soporte. • Muestra como integrar los pasos de la generación de pruebas en el proceso de desarrollo.
Inconvenientes.
• No existe versión gratuita ni de evaluación de esta herramienta. • Esta propuesta apenas profundiza en el proceso de generación, ya que solo 2 de los 12 pasos tienen relación con la generación de pruebas. • La revisión de ambigüedad se describe de una manera muy pobre, • La revisión de ambigüedad está basado en la percepción de los participantes por lo que no puede garantizarse su éxito. Lo que es ambiguo para uno puede no serlo para otro. • No se aplica ninguna técnica formal, por lo que sería prácticamente imposible desarrollarlo de manera automática con una herramienta software. • No se ofrecen referencias a aplicaciones reales. • Los ejemplos son triviales y poco clarificadores. • No queda clara la utilidad de elaborar diagramas de causa y efecto, ya que las tablas de decisión expresan la misma información que estos diagramas. Sería posible elaborar directamente las tablas. • La propuesta no detalla como tratar las dependencias entre distintos requisitos funcionales. • Esta propuesta no ofrece información sobre la mayoría de pasos que la componen. • Debido a la falta de documentación, es prácticamente imposible de aplicar con garantías de éxito sin el apoyo de su herramienta.

Dado que no ha sido posible obtener una copia de evaluación de la herramienta, no se incluye ningún ejemplo de esta propuesta.

3.9. Extended Use Case Test Design Pattern

Esta propuesta muestra cómo generar pruebas a partir de casos de uso extendidos con información adicional. El punto de partida son casos de uso descritos en lenguaje natural. Esta propuesta no exige ningún formalismo para representar los casos de uso, pero sí exige un conjunto mínimo de información.

Al final de la aplicación de esta propuesta se obtiene una tabla con todos los posibles casos de prueba para verificar cada uno de los casos de uso.

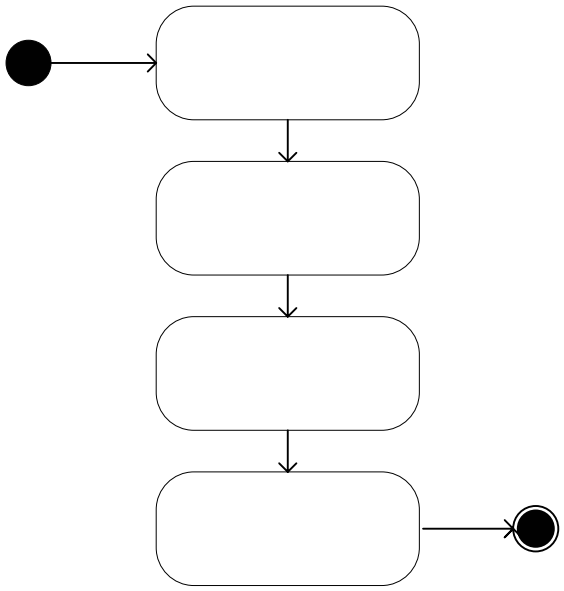
El conjunto de pasos de esta propuesta y su resultado se muestra en la tabla 36.

Tabla 36. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Identificación de variables operacionales.	Listado de variables operacionales para un caso de uso.
2	Definición del dominio de variables operacionales.	Dominio de cada una de las variables operacionales.
3	Identificación de relaciones operacionales	Lista de relaciones existentes entre las distintas variables operacionales.
4	Generación de casos de prueba.	Casos de prueba.

La ilustración 13 describe el orden de ejecución de los pasos.

Ilustración 13. Diagrama de actividades.



Un caso de uso extendido (EUC) es aquel caso de uso que contiene la siguiente información:

- Una lista completa de variables operacionales.
- Una especificación completa de restricciones de dominio para cada variable operacional.
- Una relación operacional por cada caso de uso.
- La frecuencia relativa de cada caso de uso. Este elemento es opcional.

Una variable operacional es un factor que determina cuál de las posibles secuencias de interacciones de un caso de uso se va a ejecutar en un momento concreto. Por ejemplo, si en el caso de uso se define un conjunto de interacciones según se introduzca un valor válido o inválido, ese será una variable operacional.

En el primer paso de esta propuesta, para cada variable operacional es necesario identificar la siguiente información:

- Entradas y salidas relacionadas.
- Condiciones del sistema que provoquen diferencias significativas en su comportamiento.
- Abstracciones del estado del sistema a prueba.

A continuación se identifican los dominios de dichas variables. Los dominios definen los valores válidos e inválidos para cada variable de dominio.

Una relación operacional determina dependencias entre variables operacionales de distintos casos de uso. Cada relación tiene asociada una acción del sistema que se producirá cuando todas las variables adquieran el valor adecuado. Estas relaciones se expresan mediante tablas de decisiones. Estas tablas recogen los valores que deben tener todas las variables relacionadas para que se produzca la acción esperada.

En el cuarto y último paso se generan los casos de prueba a partir de las tablas de decisiones obtenidas. Cada una de las filas de las tablas de decisiones será un caso de prueba. Esta propuesta no ofrece ningún comentario sobre la automatización del proceso de generación de pruebas. Sí, en cambio, ofrece una indicación sobre la automatización de la ejecución de las pruebas. En concreto, esta propuesta propone un proceso de grabación / reproducción de scripts.

Aunque es posible empezar a extender los casos de uso y generar pruebas desde el primer momento. Esta propuesta propone que, debido a la cantidad de revisiones que sufren los casos de uso, este proceso se posponga hasta que se cumplan las dos siguientes condiciones.

1. Los casos de uso extendidos hayan sido desarrollados y validados.

2. El sistema bajo prueba haya pasado un conjunto de pruebas de integración qué demuestren que los componentes necesarios cubren un mínimo de operatibilidad.

Esta propuesta no estudia el número de casos de prueba a desarrollar, sino cuándo parar el proceso de generación de prueba. El grado de cobertura mínimo proporcionado por cada requisito viene dado por [Srivastaba97] en la fórmula que se recoge en la tabla 37.

Tabla 37. Cálculo del grado de cobertura.

(Numero de características implementadas / Numero de características requeridas) X (Componentes totales probados / Numero total de componentes) X 100

La ventaja principal de esta propuesta es indicar claramente cuando debe comenzar el proceso de prueba y proponer el estándar IEEE 982.1 [Srivastaba97] para evaluar la cobertura de las pruebas. Mediante esta evaluación se puede decidir cuando detener el proceso de generación de pruebas. Sin embargo el trabajar con casos de uso expresados en lenguaje natural dificulta la sistematización y automatización de esta propuesta. La descripción de la generación de casos de prueba es muy pobre y no ofrece ningún ejemplo ni referencias a aplicaciones prácticas. En la tabla 38 se presentan de manera resumida las principales ventajas e inconvenientes de esta propuesta.

Tabla 38. Resumen de ventajas e inconvenientes.

Ventajas.
• Esta propuesta ofrece una referencia clara de cuando puede comenzar el proceso de generación de pruebas. • Incluye una referencia al estándar IEEE 982.1 como referencia para evaluar la cobertura de las pruebas generadas.
Inconvenientes.
• La propuesta describe la información que debe aparecer en un EUC, pero no dice como añadirla un caso de uso. No da ninguna notación, plantilla o extensión a UML. • Está completamente basada en lenguaje natural, por lo que sería difícil desarrollarla de manera automática con una herramienta software. • No se ofrecen referencias a aplicaciones reales. • No detalla qué se obtiene al final de su aplicación ni cómo generar pruebas ejecutables.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.10. Software Requirements and Acceptance Testing.

Este trabajo [Hsia97] está centrado en la generación de pruebas de aceptación. Sin embargo, la definición que hace este trabajo (tabla 39) de prueba de aceptación coincide con la definición de prueba del sistema recogida en la tabla 1. Por este motivo incluimos esta propuesta en nuestro estudio.

Tabla 39. Definición de prueba de aceptación.

Las pruebas de aceptación consisten en comprara un sistema software con sus requisitos iniciales y con las necesidades de sus usuarios finales.

Esta propuesta se divide en dos bloques, en el primero se realiza un resumen del proceso de elicitación de requisitos. Este proceso se expone con más detalle en [Hsia94]. El segundo bloque consiste en la construcción de pruebas de aceptación a partir de los escenarios obtenidos en el bloque anterior.

Puesto que esta propuesta incluye una serie de pasos para la elicitación de requisitos, el punto de partida son las necesidades de los futuros usuarios del sistema, al igual que SCENT. El objetivo de esta propuesta es construir un modelo de prueba para la generación de pruebas de aceptación, y generar pruebas a partir de dicho modelo.

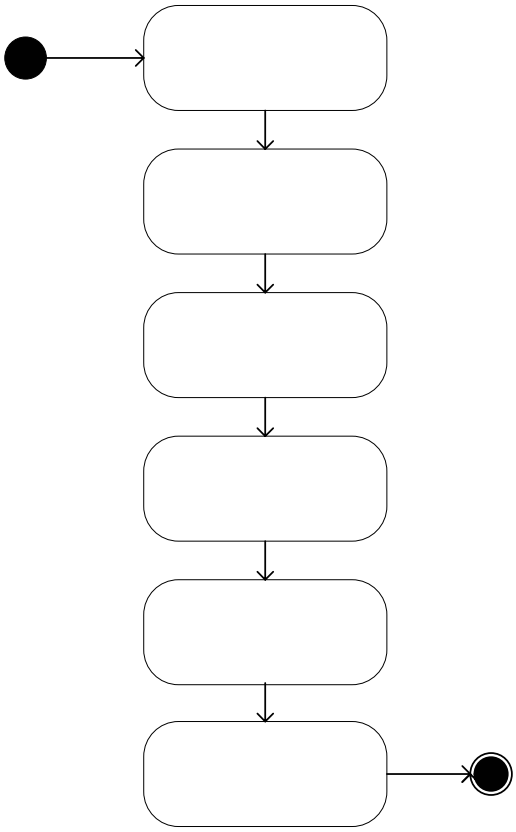
El proceso de elicitación de requisitos consta de 6 actividades que se desarrollan de manera secuencial. La tabla 40 muestra un resumen de estos pasos.

Tabla 40. Pasos para la obtención de escenarios.

Paso	Descripción	Resultado
1	Elicitación de escenarios.	Escenarios expresados mediante árboles de escenarios.
2	Formalización de escenarios.	Una gramática por cada árbol identificado en el paso 1.
3	Verificación de escenarios.	Escenarios formales abstractos.
4	Generación de escenarios	Escenarios concretos a partir de los escenarios abstractos.
5	Generación de prototipos	Prototipos del sistema a partir de los escenarios del paso anterior.
6	Validación de escenarios	Escenarios validados.

La ilustración 14 describe el orden de ejecución de los pasos.

Ilustración 14. Diagrama de actividades.



Esta propuesta define escenario como una descripción proporcionada por un usuario de como desea utilizar el sistema software. Este escenario consiste en una secuencia de eventos entre un usuario y un sistema. Para la elicitación de requisitos se representa el escenario de uso mediante una notación gráfica, tomada de [Hsia94], llamada árboles de escenarios. Estos árboles representan el escenario desde el punto de vista de un usuario concreto del sistema.

Un ejemplo de árbol de escenarios y de su gramática asociada se muestra en la ilustración 15 y en la tabla 41.

Ilustración 15. Árbol de escenario para el acceso al sistema.

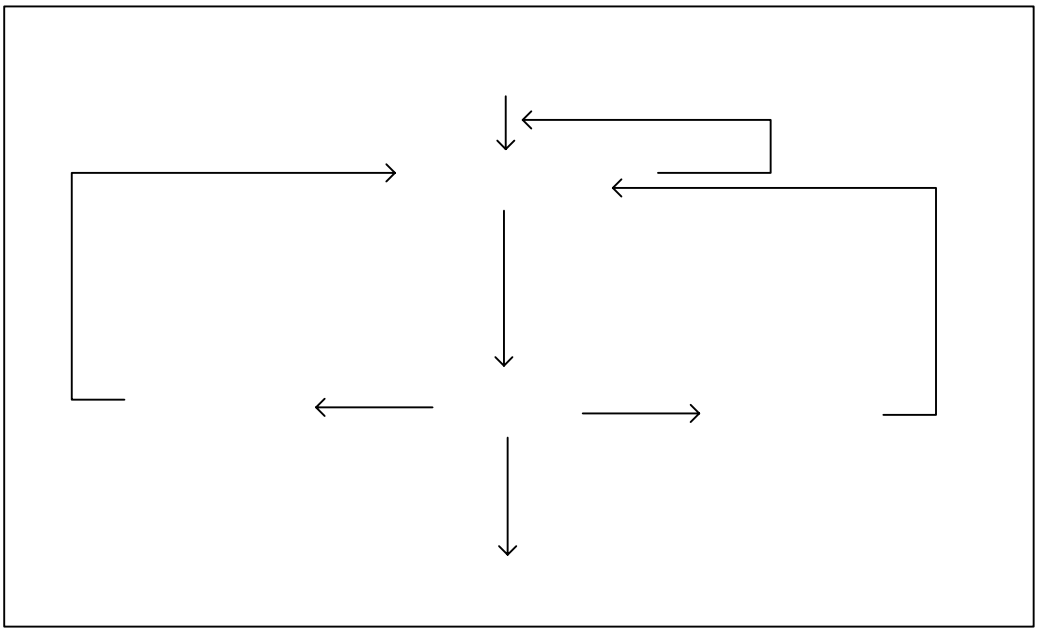


Tabla 41. Gramática generada a partir del árbol de escenario.

Eventos = { Acceso, Nombre o contraseña en blanco, Entrar autentificación, Nombre inexistente, Contraseña no válida, Autentificación válida }

Estados = { Mostrar formulario de autentificación, Comprobar autentificación, Mensaje de error }

Estado inicial = Usuario

Reglas = { Usuario -> Acceso (Mostrar formulario de autentificación),

Mostrar formulario de autentificación -> Nombre o contraseña en blanco (Mostrar formulario de autentificación) | Entrar autentificación (Comprobar autentificación),

Entrar autentificación -> Autentificación válida | Nombre inexistente (Mensaje de error) | Contraseña no valida (Mensaje de error),

Mensaje de error -> Acceso (Mostrar formulario de autentificación), }

Acceso

El objetivo del segundo bloque es elaborar un modelo de pruebas de sistema y aceptación a partir de los escenarios del bloque anterior. Este modelo permitirá probar el comportamiento externo del sistema expresado en los escenarios. Este modelo estará compuesto de tres submodelos que se detallan en la tabla 42.

Tabla 42. Submodelos de prueba.

Modelo	Nombre	Descripción
1	Vista del usuario.	Describe como un conjunto de usuarios utilizaran el sistema, es decir, las entradas y salidas del sistema
2	Interfaz externa del sistema.	Modela el comportamiento externo del sistema.

Mensaje de

3	Vista externa del sistema.	Modela el comportamiento del sistema combinando la vista del usuario y la interfaz externa del sistema.

El segundo bloque comienza a partir de los escenarios generados en el primer bloque. Este segundo bloque está compuesto por tres pasos. Estos pasos se describen en la tabla 43

Tabla 43. Pasos para la generación de escenarios de prueba.

Paso	Descripción	Resultado
1	Elicitación y especificación de escenarios.	Vistas de usuario e interfaces externas.
2	Formalización del modelo de pruebas.	Conjunto de máquinas de estados.
3	Verificación del modelo de pruebas.	Máquinas de estados verificadas.

A pesar de tener el mismo nombre, el paso de elicitación de escenarios en el segundo bloque es distinto del paso en el primer bloque. En este caso el objetivo es identificar el conjunto de vistas del usuario y las interfaces externas a partir de los escenarios del bloque anterior. Estas vistas de usuarios e interfaces se expresan mediante árboles de escenarios similares al de la ilustración 15. A continuación dichos árboles se traducen a máquinas de estados. Todas las máquinas de estados se combinan en una sola que representa el modelo de prueba del sistema. Finalmente el modelo de prueba del sistema se verifica para corregir inconsistencias, redundancias u omisiones.

Para generar pruebas a partir de estos modelos se construyen matrices de caminos para cada una de las máquinas virtuales. Estas matrices son cuadras y de dimensión igual al número de estados de la máquina virtual. Esta propuesta, sin embargo, no entra en detalles sobre como obtener pruebas y remite a [Hsia94].

Una aplicación de esta propuesta en dos proyectos reales se documenta en [Hsia95].

No hemos encontrado ninguna ventaja de esta propuesta que destaque sobre las demás, lo cual es justificable dada su antigüedad. Su principal inconveniente es que es una puesta al día de una propuesta de 1994. La definición de sus términos y procesos es escueta y ambigüa, no explicando muchos de los conceptos, y su notación de árboles no aporta nada respecto a los diagramas propuestos en UML. En la tabla 42 se resumen sus ventajas e inconvenientes.

Tabla 44. Resumen de ventajas e inconvenientes.

Ventajas.
• El trabajo incluye un apartado sobre la situación de los procesos de prueba en la industria y el motivo de que no estén presentes.
Inconvenientes.
• No se incluyen referencias a proyectos reales donde se haya aplicado esta propuesta • No indica como implementar las pruebas. • Es una propuesta antigua (1997) por lo cual su análisis del estado del arte ya está desfasado. Además, su principal referencia es de 1994, por lo que, en conjunto, tiene más de 10 años de antigüedad. • Su definición de escenario es pobre, ya que no contempla la interacción del sistema con otros sistemas, sino solo con usuarios. Esto es comprensible al estar orientada a pruebas de aceptación. • Es una propuesta ambigua y con términos contradictorios, por ejemplo en el proceso de elicitación de requisitos después de elicitar, formalizar y verificar escenarios, se generan, lo cual es muy confuso. • Un árbol de escenario no aporta nada que no aporte un diagrama de flujo o diagrama de estados. • La propuesta no hace ninguna referencia a su automatización o a herramientas de soporte. • Es poco automatizable, no es posible construir las reglas de manera automática a partir del árbol. • Las reglas para construir la gramática no están claras. No son las reglas que se utilizan habitualmente en la gramáticas de lenguajes de programación (por ejemplo mezclar transiciones con estados. • Constantemente hace referencias a [Hsia94] por lo que más parece un resumen de dicho trabajo, más que una propuesta que aporte algo nuevo. De hecho, la referencia al caso real de aplicación es del año 1995,

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.11. Use Case Derived Test Cases.

El punto de partida de esta propuesta [Wood02] es un caso de uso descrito en lenguaje natural. Esta propuesta indica la información que debe tener un caso de uso. Esta información se recoge en la tabla 45.

Tabla 45. Información necesaria para generar casos de prueba.

Nombre Breve descripción

Requisitos SRS. Precondiciones y poscondiciones. Flujo de eventos.

Al final de esta propuesta se obtiene un conjunto de casos de prueba descritos en lenguaje natural. Cada caso de prueba define las acciones a realizar en el sistema y las verificaciones necesarias. En la tabla 46 se muestran dos ejemplos de casos de prueba para verificar un sistema de apertura de puerta mediante tarjeta de seguridad.

Tabla 46. Ejemplos de casos de prueba generados por esta propuesta.

Condición de prueba 1: Tarjeta de empleado válida • Introducir tarjeta. • Verificar que la puerta está abierta • Entrar en el edificio. • Verificar que la puerta está cerrada Condición de prueba 2: La tarjeta no puede leerse. • Introducir tarjeta. • Verificar que el evento se registra.

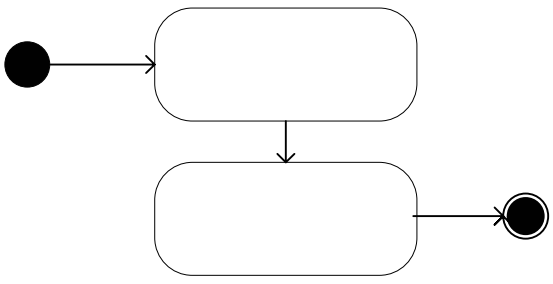
El conjunto de pasos de esta propuesta y su resultado se muestra en la tabla 47.

Tabla 47. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Identificación de caminos de ejecución.	Todos los caminos diferentes a partir de un caso de uso.
2	Diseño de casos de prueba	Casos de prueba descritos en lenguaje natural.

La ilustración 16 describe el orden de ejecución de los pasos.

Ilustración 16. Diagrama de actividades.



En el primer paso se identifican todos los caminos diferentes en el flujo de eventos recogidos en el caso de uso. Cada uno de estos caminos se transformará en un caso de prueba en el segundo paso.

Esta propuesta no supone ninguna ventaja respecto a las propuestas ya estudiadas. Sus principales inconvenientes es su falta de precisión en la documentación y su nula escalabilidad. En la tabla 48 se resumen sus ventajas e inconvenientes.

Tabla 48. Resumen de ventajas e inconvenientes.

Ventajas.
• Ninguna.
Inconvenientes.
• Esta propuesta no describe con precisión cuando un camino es diferente a otro camino. • Esta propuesta no define qué es un requisito SRS. • No permite seleccionar los casos de prueba, por lo que esta propuesta no es aplicable en sistemas grandes. • Este trabajo solo cuenta con tres referencias, lo que revela un nivel de investigación académica muy pobre. • La referencia más actual data de 1997, lo cual es una medida de la poca calidad de este trabajo, ya que desde 1997 hasta el año 2002 en que se publica este trabajo han aparecido otros artículos y libros muy relevantes, como [Binder00]. • Al describir los casos en lenguaje natural de una manera informal, imposibilita automatizar la generación de código de pruebas a partir de ellos.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.12. A Model-based Approach to Improve System Testing of Interactive Applications

Esta propuesta (TDEUML) está basada en los trabajos sobre generación de casos de prueba desarrollados por la empresa Siemens desde 1992, año de aparición de la primera versión de su herramienta TDE centrada en las pruebas unitarias [Ruder04-2].

El punto de partida de esta propuesta [Ruder04] es la documentación de casos de uso en lenguaje natural. Al final de la aplicación de esta propuesta se obtiene un conjunto de pruebas de un sistema a través de su interfaz gráfica. Estas pruebas son ejecutables en herramientas que permitan interactuar con una interfaz gráfica a partir de las instrucciones almacenadas en un script de prueba.

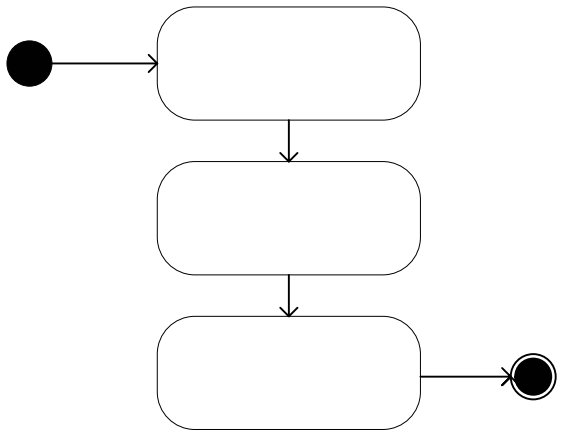
El proceso de generación de casos de prueba se puede resumir en tres pasos. Estos pasos se describen en la tabla 49.

Tabla 49. Conjunto de pasos para la generación de pruebas.

Paso	Descripción	Resultado
1	Modelado del comportamiento del sistema	Diagramas de actividades con requisitos de prueba.
2	Diseño de casos de prueba	Conjunto de scripts de prueba.
3	Ejecución automática de casos de prueba	Conjunto de scripts de prueba refinados.

La ilustración 17 describe el orden de ejecución de los pasos.

Ilustración 17. Diagrama de actividades.



En el paso de modelado del comportamiento del sistema se construye un modelo de comportamiento del sistema. Esta propuesta propone los diagramas de actividades de UML para describir el comportamiento interno de cada caso de uso. Una vez construidos los diagramas de actividades, se completan con la especificación de requisitos de prueba. Esta propuesta define los requisitos de prueba como un conjunto de estereotipos que servirán de guía a la herramienta para indicarle como debe interpretar cada actividad. Cada actividad debe ser anotada con uno de los estereotipos siguientes: <<UserAction>>, <<SystemResponse>> o <<Include>>. Mediante estos estereotipos es posible indicar si es una actividad del usuario o del sistema o está descrita mediante otro diagrama de actividades. Estos estereotipos serán interpretados por el generador de pruebas a la hora de construir pruebas ejecutables.

A partir de los diagramas de actividades y sus estereotipos se aplica el segundo paso de la tabla 49. En primer lugar se construye el diseño de pruebas. Esta propuesta define el diseño de pruebas como la descripción de los diagramas de actividades en

lenguaje TSL [Balcer90]. A partir de este diseño en lenguaje TSL es posible generar guiones o scripts de prueba. Esta propuesta propone la herramienta TDE [Balcer90] para realizar la traducción de TSL a scripts de prueba. El resultado de la aplicación de esta herramienta es un conjunto de guiones de pruebas ejecutables en una herramienta de verificación de interfaces gráficas.

El tercer paso consiste en la ejecución de estos guiones sobre el sistema en pruebas. Las primeras ejecuciones pueden servir también para refinar y mejorar el conjunto de scripts de pruebas generado.

La principal ventaja de esta propuesta es que describe como generar pruebas ejecutables. Sus principales inconvenientes son que no define con el suficiente detalle como obtener los diagramas de actividades a partir de los casos de uso ni el criterio asignar los estereotipos a cada actividad. Tampoco especifica si el paso de diagramas de actividades a TSL es un proceso manual o puede automatizarse. Un resumen de las ventajas e inconvenientes de esta propuesta se muestra en la tabla 48.

Tabla 50. Resumen de ventajas e inconvenientes.

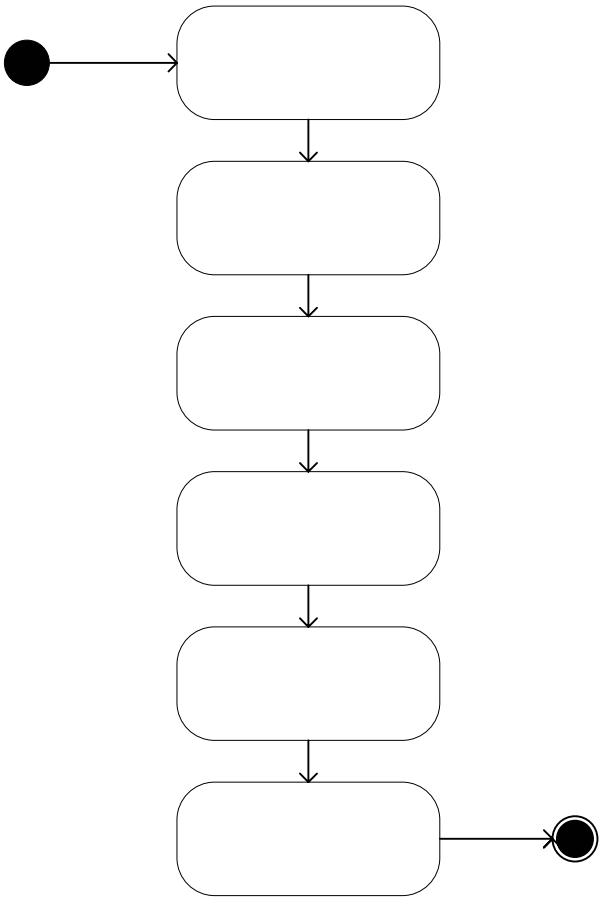
Ventajas.
• Indica como generar pruebas ejecutables sobre el sistema. • El uso de estereotipos permite añadir información necesaria para la automatización del proceso de generación de pruebas respetando el estándar UML. • El uso de un lenguaje formal facilita también la automatización.
Inconvenientes.
• Se centra en interfaces gráficas, por lo que no permite generar casos de prueba para aplicar a otro tipo de interfaces externas de un sistema que permitan la comunicación, no con humanos, sino con otros sistemas. • No describe con detalle como construir los diagramas de actividades, • Tampoco describe si es necesario un diagrama de actividades por cada caso de uso o un único diagrama para todo el sistema. • No explica cómo incluir en los diagramas de actividades dependencias entre casos de uso. • No se ha encontrado ninguna versión de evaluación o con licencia educativa para evaluar la herramienta.

Un ejemplo de esta propuesta se puede encontrar en el punto de casos prácticos.

3.13. RETNA

Requirements to Testing in a Natural Way (RETNA) es un analizador de requisitos que, según sus autores, puede ser utilizado tanto para generar pruebas de caja negra como de caja blanca [Boddu04]. Las actividades de esta propuesta se muestran en la ilustración 18.

Ilustración 18. Diagrama de actividades.



RETNA se divide en dos grandes bloques, en el primero (actividades de la 1 a la 5) se analizan los requisitos en el lenguaje natural y, a partir de ellos, se genera una máquina de estados finitos expresada en lenguaje MONA [Henriksen95]. En el segundo (actividad 6) se generan casos de prueba a partir del modelo del bloque anterior.

En la primera actividad los requisitos se redactan en un párrafo de no más de 399 palabras. A continuación se identifica cada una de las frases del texto. A partir de cada frase se genera un árbol anotado que incluye los términos de cada frase junto con

etiquetas que indican su función. Un ejemplo de árbol anotado se muestra a continuación.

Si un tren está en una estación, su velocidad debe ser menor de 30

```
(S1 (S ('' ')
       ('' ')
       (SBAR (IN si)
         (S (NP (DT un) (NN tren))
(VP (AUX está) (PP (IN en) (NP (DT una)
      (NN estación))))))
      (ADVP (RB entonces))
      (NP (PRF $ su) (NN velocidad))
      (VP (MD debe)
        (VP (AUX ser)
          (ADJP (ADJP (JJR menor))
           (PP (IN que)
               (NN 30) ))))
```

Sin embargo, las anotaciones no son capaces de clarificar entre verbos, sujetos y otros elementos. Por este motivo a continuación se identifica el significado semántico de cada elemento del árbol. Como resultado se obtiene una estructura de representación del discurso (Discourse Representation Structure o DRS [Blackburn94]). Esta estructura se refina para eliminar indefiniciones y ambigüedades. Después el DRS se traduce a MONA. Esto se realiza de manera automática. El resultado de MONA es una máquina de estados finitos.

En el segundo bloque se generan casos de prueba recorriendo la máquina de estados. Es posible aplicar distintos tipos de recorridos sobre la máquina de estados para obtener un conjunto de pruebas. Para evitar la explosión de estados RETNA propone que se pregunte al usuario por los requisitos más críticos y generar sólo casos de prueba para ellos. La máquina de estados también puede utilizarse para construir un test oracle. No se incluye ningún ejemplo de las pruebas generadas en la documentación de RETNA.

La principal ventaja de esta propuesta es proporcionar un proceso completo para procesar requisitos en lenguaje natural. Mediante este procesamiento es posible automatizar todo el proceso de generación de pruebas del sistema. Sus principales inconvenientes son la falta de detalle con que se describe, la falta de ejemplos, y la no disponibilidad de las herramientas. Esta propuesta tampoco describe los criterios para recorrer las máquinas de estados generadas ni los criterios para seleccionar un subconjunto de estas máquinas para evitar una explosión de casos de prueba. Tampoco indica si las pruebas generadas cubren todos los requisitos expresados en el párrafo en lenguaje natural, ni la correspondencia de las máquinas de estados con los requisitos, ni muestra ningún ejemplo de las pruebas generadas. Un resumen de estas ventajas e inconvenientes se enumeran en la tabla 51.

Tabla 51. Resumen de ventajas e inconvenientes.

Ventajas.
• Proceso completamente automatizable para la obtención de un conjunto de casos de prueba a partir de un texto en lenguaje natural.
Inconvenientes.
• No se detallan los resultados obtenidos. • No se muestra ningún ejemplo de prueba. • No describe como obtener pruebas a partir de las máquinas de estados generadas. • No indica si la propuesta tiene en cuenta dependencias entre distintos requisitos. • Tampoco indica como trabajar cuando lodos los requisitos superan el límite de caracteres.

RETNA está compuesta por un conjunto de herramientas para realizar las actividades mostradas en la ilustración 18. Algunas de esas herramientas son de libre acceso, pero otras han sido desarrolladas por los autores (como los scripts para el procesamiento del lenguaje natural) y no ha sido posible obtener una copia, por lo que no se ha desarrollado ningún caso práctico a partir de esta propuesta.

3.14. Otras propuestas.

A lo largo la elaboración de este informe de investigación, se han estudiado otras propuestas, trabajos y herramientas que finalmente no han sido incluidas en el estudio comparativo. En los siguientes puntos se comentan de manera resumida algunos de

estos trabajos. En cada uno de dichos trabajos se exponen los motivos por los que no ha sido incluido en el análisis comparativo.

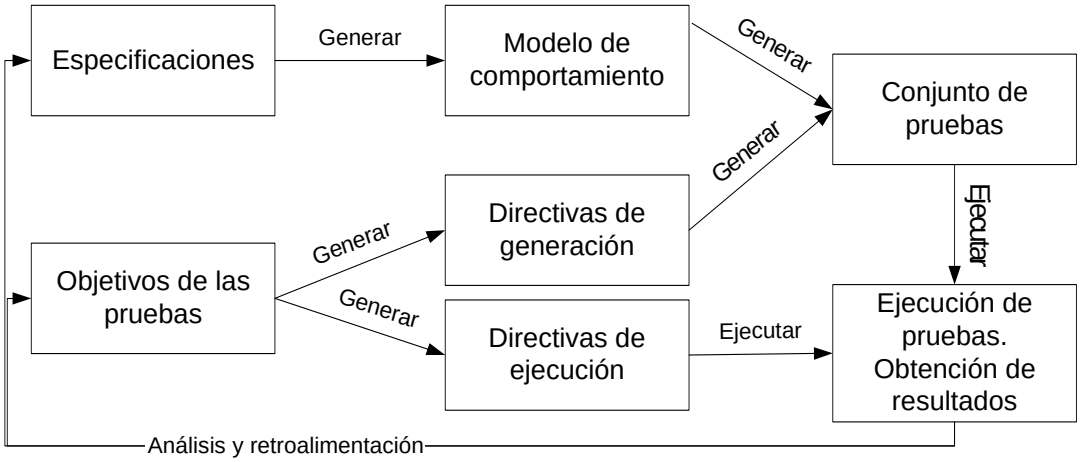
3.13.1. AGEDIS.

AGEDIS [Hartman04], [Cavarra04] y [Craggs03] fue un proyecto de investigación financiado por la Unión Europea que concluyó a principios del año 2.004. Su principal objetivo ha sido el desarrollo de una metodología del mismo nombre y 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, se obtienen mejores resultados con sistemas de control, como protocolos de comunicaciones, que con sistemas de transformación de información, como compiladores.

AGEDIS se centra en dos productos: un modelo del sistema escrito en un lenguaje de modelado y en diagramas UML de clases y estados que va a permitir la generación automática del conjunto de pruebas, y un conjunto de objetos de casos de prueba que sirven de enlace entre el modelo del sistema y la implementación de dicho sistema, permitiendo ejecutar las pruebas tanto en el modelo como en la implantación y comparando los resultados esperados, los del modelo, con los obtenidos, los del sistema.

La metodología desarrollada expone un proceso iterativo de seis pasos, como se muestra en la ilustración 19. 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 donde cada clase tiene asignada un diagrama UML de estados 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. Por último se analizan los resultados y se repiten los pasos hasta que se alcanzan los objetivos deseados.

Ilustración 19. Descripción del proceso de obtención y ejecución de casos de prueba.



Se ha comprobado, mediante sus casos prácticos [Craggs03], como AGEDIS puede ser también efectiva sin necesidad del conjunto completo de requisitos, mediante el modelado y prueba de un sistema que recoja un subconjunto de los requisitos del sistema.

AGEDIS no ha sido incluido en el estudio porque, como uno de sus principales autores nos ha comentado, AGEDIS tiene poco que ver con la especificación funcional de un sistema. El punto de partida de AGEDIS es, en cambio, un conjunto de diagramas de clases y de estados que modelan el comportamiento del sistema, no indicando un proceso para generar dichos modelos a partir de una especificación funcional.

3.13.2. Verificación con XML.

Este trabajo [Vazquez01] presenta un marco de trabajo para la verificación automática de programas en entornos de sistemas orientados a objetos.

El núcleo del sistema lo constituye un repositorio de clases descritas mediante documentos XML. Un parser o analizador sintáctico analiza el código fuente, extrae de él la información relevante del programa y alimenta el repositorio.

El lenguaje XML es utilizado como metalenguaje para la creación de un árbol de sintaxis abstracta independiente del lenguaje de programación utilizado. El repositorio sirve para la creación de casos de prueba.

Las ventajas de utilizar XML como metalenguaje se pueden observar en la extensibilidad del entorno de trabajo: basta con añadir un parser para otro lenguaje y el sistema completo será utilizable para dicho lenguaje. No será necesario modificar las herramientas creadas para el sistema. Es más, con un parser UML es posible subir hasta

la fase de diseño, anticipando la creación de casos de prueba a dicha etapa del ciclo de vida.

Sin embargo, según sus propios autores, esta opción no ha sido desarrollada ni se prevé desarrollarla en el futuro. Por este motivo, este trabajo no ha sido incluido dentro de las propuestas estudiadas en este análisis.

3.13.3. Automatic Test Case Generation for RAISE

En este trabajo [Dan02] se desarrolla un método sistemático para la derivación de casos de prueba a partir de las especificaciones RSL de un sistema software. RSL [RAISE92], [RAISE95] son las siglas de Raise Specification Language, un lenguaje para la especificación formal de sistemas software.

Los trabajos basados en RAISE no han sido tenidos en cuenta en el estudio comparativo desarrollado en este trabajo, ya que, para este trabajo se ha tomado como criterio de selección la generación de pruebas a partir de la especificación funcional expresada en lenguaje natural. Sin embargo los trabajos con RAISE generan pruebas a partir de la especificación formal redactada en este lenguaje. Para incluir esta propuesta sería necesario un paso previo consistente en traducir la especificación funcional a lenguaje RAISE, lo cual no es contemplado en ninguno de los trabajos reseñados.

3.13.4. Trabajos de investigación del profesor Jeff Offut

Este investigador ha desarrollado varios trabajos en el campo de las pruebas y, más en concreto, en el campo de la generación de pruebas del sistema., Sin embargo, ninguno de sus trabajos encajan en el perfil descrito en la introducción de la sección 3. Por este motivo ninguno de ellos ha sido incluido en el análisis comparativo de este trabajo. A continuación, a modo de ejemplo, se referencian algunos de sus trabajos y los motivos por los que no han sido incluidos.

Modeling and Testing Web-based Applications

Este trabajo [Wu02] propone un método para modelar una aplicación web a partir de las páginas que las componen, en lugar de las especificación del sistema.

Por este motivo no ha sido incluido en la comparativa.

Testing Web Applications by Modeling with FSMs

Este trabajo [Andrews05] propone cómo modelar una aplicación web en base a máquinas de estados finitos y cómo generar pruebas a partir de dichas máquinas.

En este trabajo se mezcla el concepto de la funcionalidad con el concepto de la navegabilidad en sistemas web. Sin embargo, ya existen trabajos como [Escalona04] que, a nivel de requisitos, muestran la conveniencia de separar estos dos conceptos.

El trabajo Testing Web Applications by Modeling with FSMs se basa en dividir la aplicación web en clusters, sin embargo no se define con precisión como tiene que estar especificada la aplicación web, ni si es posible aplicar este proceso sólo con la especificación funcional de la aplicación (y, en ese caso, faltaría indicar las características o información de dicha especificación formal) o si solo puede aplicarse una vez que se tiene completamente construida la aplicación web. En el ejemplo incluido en el trabajo, este modelado se realiza en la fase de diseño, a partir de una descripción informal del sistema.

Por estos motivos no ha sido incluido en el análisis presentado en este trabajo.

Generating Tests from UML Specications

Este trabajo [Offutt03] muestra cómo generar casos de prueba a partir de diagramas de estados definidos en UML. Este trabajo es muy referenciado entre las propuestas estudiadas. No se ha incluido en el análisis comparativo porque no describe cómo obtener los diagramas de estados a partir de la especificación funcional del sistema en lenguaje natural.

4. Un caso práctico.

En esta sección se describe un conjunto de casos de uso y genera un conjunto de pruebas de sistema aplicando las propuestas descritas en la sección anterior.

Como se ha comentado en el punto 3.8, no se incluye ningún ejemplo de la propuesta Requirements Based Testing.

4.1. Descripción de los casos de uso.

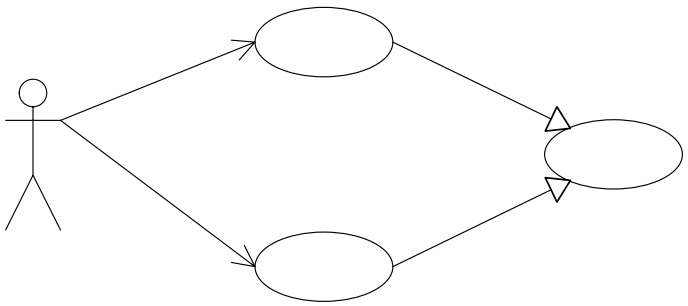
El ejemplo está basado en un sistema de préstamo bibliotecario. Este sistema se describe con más detalle en las propuestas SCENT y TOTEM. La tabla 52 muestra los casos de uso que se van a utilizar en este caso práctico

Tabla 52. Casos de uso seleccionados de un sistema de préstamo bibliotecario.

UC-01	Acceso al sistema.
UC-02	Préstamo de libros.
UC-03	Devolución de libros.

El diagrama de casos de uso se muestra en la ilustración 20.

Ilustración 20. Diagrama de casos de uso.



A continuación, en las tablas 53, 54 y 55, se describen los casos de uso mediante el modelo de plantillas propuesto en [Escalona04]. Los elementos definidos en el modelo de plantilla que no tienen sentido en este ejemplo han sido omitidos de las plantillas.

Tabla 53. Caso de uso "Entrada en el sistema".

UC-01	**Acceso al 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 recibe la conexión y muestra una página donde pide nombre y contraseña
	3	El usuario rellena el formulario introduciendo su nombre y pulsa el botón entrar
	4	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 54. Caso de uso "Préstamo de libros".

UC-02	**Préstamo de libros**	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario quiera recibir libros en préstamo	
Precondición	Usuario identificado.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario conecta con la página de préstamo de libros.
	2	El sistema solicita la identificación del libro

	3	El usuario introduce la identificación del libro que quiere obtener en préstamo
	4	El sistema verifica que el libro está disponible para préstamo y que el usuario puede tomarlo en préstamo y muestra un mensaje de confirmación.
Excepción	**Paso**	**Acción**
	3	Si el usuario introduce una identificación en blanco el sistema vuelve a solicitar la identificación.
	4	Si el libro no existe se muestra un mensaje de error y termina el caso de uso.
	4	Si el libro no está disponible para préstamo se muestra un mensaje de error y termina el caso de uso.
	4	Si el usuario no puede solicitar más libros se muestra un mensaje de error y termina el caso de uso.
Rendimiento	**Paso**	**Cuota de tiempo**
	3	1 segundos.

Tabla 55. Caso de uso "Devolución de libros".

UC-03	**Devolución de libros**	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario quiera devolver libros en préstamo	
Precondición	Usuario identificado.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario conecta con la página de devolución de libros.
	2	El sistema muestra los libros en préstamo por el usuario
	3	El usuario pulsa la opción para devolver los libros.
	4	El sistema muestra un mensaje de confirmación.
Excepción	**Paso**	**Acción**
	2	Si el usuario no tiene libros el préstamo el sistema muestra un mensaje y termina el caso de uso.

Rendimiento	**Paso**	**Cuota de tiempo**
	2	2 segundos.

Existe una dependencia entre los casos de uso 2 y 3 y el caso de uso 1, de manera que los casos de uso 2 y 3 no pueden ejecutarse si no se ejecuta también el caso 1. También existe una dependencia temporal, ya que el caso 3 debe ejecutarse después del 2, pero sin que transcurra un tiempo mínimo entre ambos.

En los siguientes puntos, se aplicarán cada una de las propuestas analizadas en este trabajo sobre estos casos de uso para la obtención de un conjunto de pruebas de sistema.

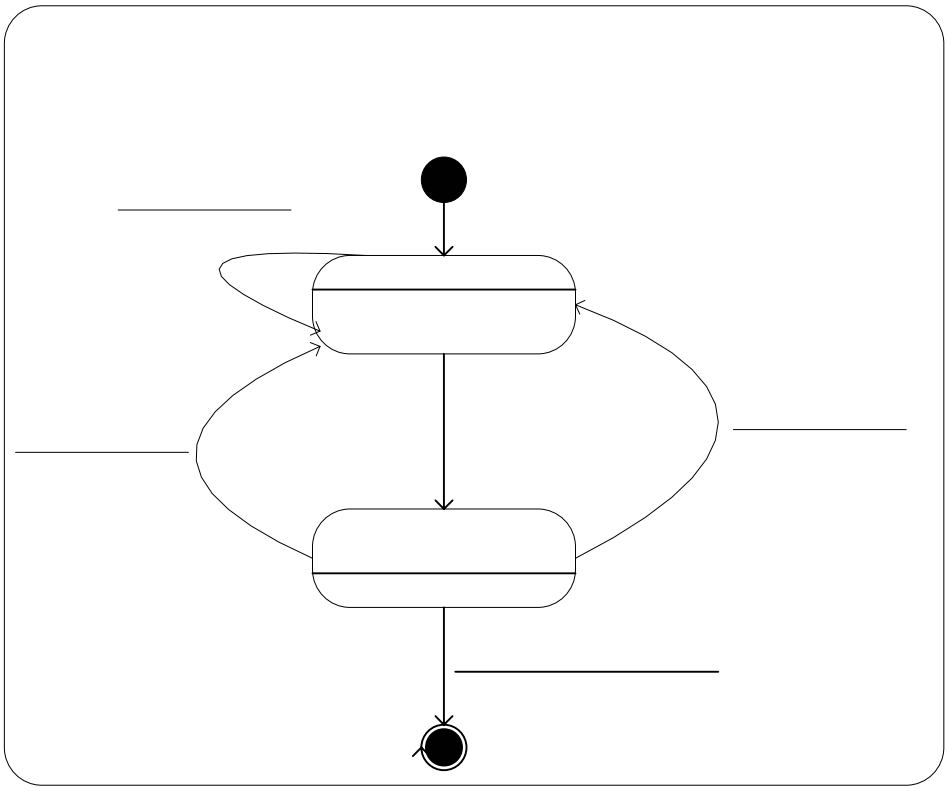
4.2. Generación del conjunto de pruebas aplicando Automated Test Case Generation from Dynamic Models.

Esta propuesta sí es capaz de generar casos de prueba que contemplen la dependencia existente entre los casos de uso 2 y 3 y el caso de uso 1, sin embargo no es capaz de generar pruebas que involucren secuencias de casos de uso, por lo que solo aplicaremos esta propuesta a los casos de uso 1 y 2.

El primer paso consiste en construir un diagrama de estados a partir de la descripción del caso de uso. En primer lugar se identifica el camino principal del caso de uso. Cada mensaje entre el sistema y los actores será una transición

Todo el diagrama de estados se encapsula en un estado para su posterior reutilización. El diagrama de estados correspondiente al caso de uso 1 se muestra en la ilustración 21.

Ilustración 21. Diagrama de estados de acceso al sistema.



A continuación se construye el diagrama de estados del caso 2. Como el caso de uso 1 está incluido en el caso de uso 2, el diagrama de estados anterior también estará incluido en el diagrama de estados del caso 2. El diagrama de estados resultante se muestra en la ilustración 22.

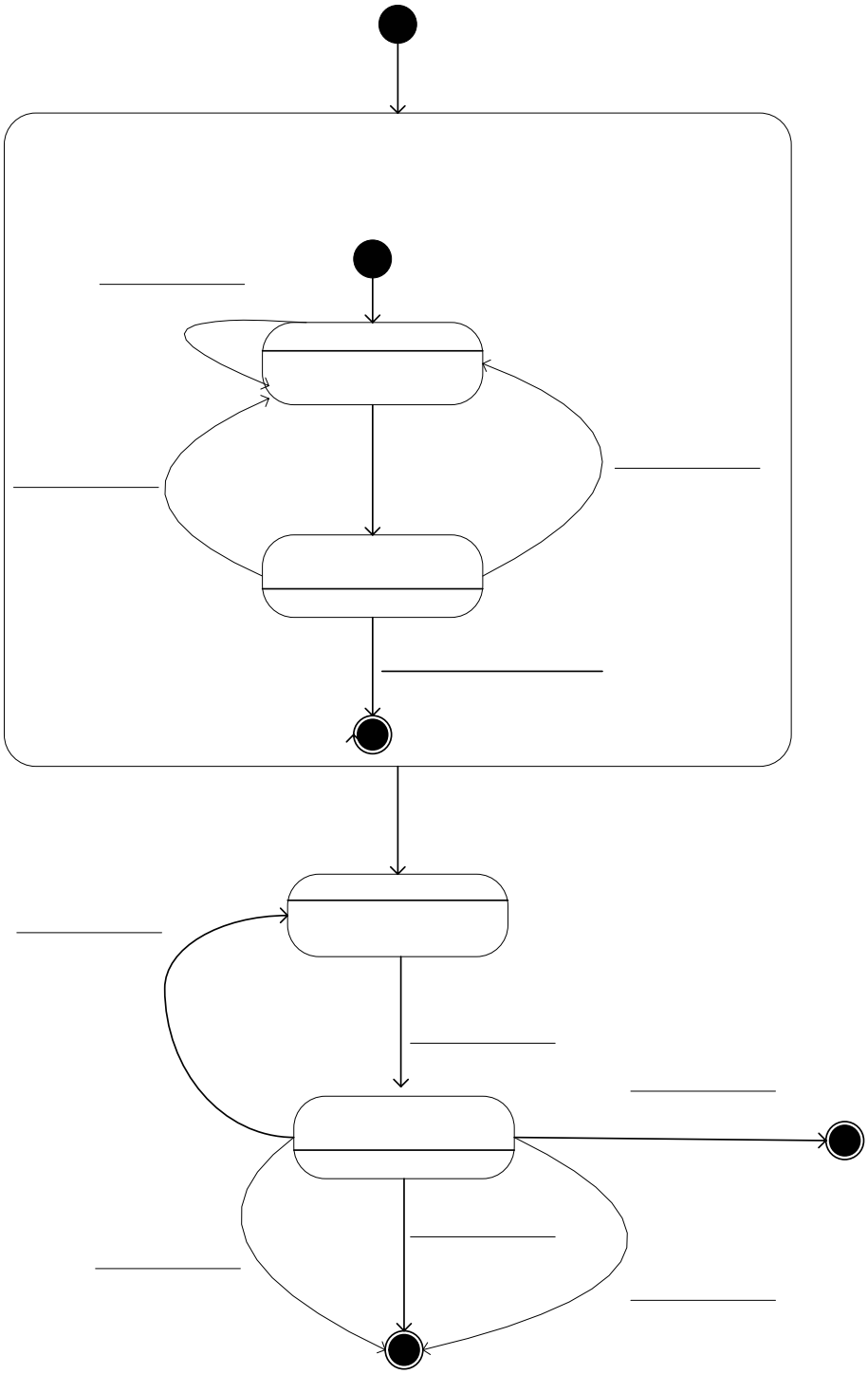
Una vez generado el diagrama de estados, es necesario extraer los operadores según la notación STRIPS. Según la notación STRIPS, un operador O es una terna de proposiciones tal y como se muestra a continuación.

O = { Precondiciones, Adicciones, Sustracciones }

Recordemos que precondiciones son el conjunto de proposiciones que deben existir para aplicar el operador, adicciones son las nuevas proposiciones que se añaden al aplicar el operador y sustracciones son las proposiciones que se eliminan una vez aplicado el operador.

Nombre o c
en bla
Mensaje
Nombre
inexistente
Mensaje de error

Ilustración 22. Diagrama de estados de préstamo de libro.



Para describir los estado se van a utilizar una serie de predicados auxiliares. El primero, in(t), indica que el estado actual del sistema es t. El segundo, log_state(t) indica que el estado t fue el estado del sistema en algún momento pasado. Por cada transición en el diagrama de estados se va a generar una proposición log_trans(t) que indica que la transición t ha sido utilizada en algún instante en el pasado.

La lista completa de operadores obtenidos a partir del diagrama de la ilustración 22 se muestra en la tabla 56.

Tabla 56. Conjunto de operadores.

Operador	Precondiciones	Addicciones	Sustracciones
O0	In(Inicial)	Log_state(Inicial) Log_trans(t0) In(s0)	In(Inicial)
O1	In(s0)	Log_state(s0) Log_trans(t1) In(s0)	< vacío >
O2	In(s0)	Log_state(s0) Log_trans(t2) In(s1)	In(s0)
O3	In(s1)	Log_state(s1) Log_trans(t3) In(s1)	< vacío >
O4	In(s1)	Log_state(s1) Log_trans(t4) In(s1)	< vacío >
O5	In(s1)	Log_state(s1) Log_trans(t5) In(s2)	In(s1)
O6	In(s2)	Log_state(s2) Log_trans(t6) In(s3)	In(s2)
O7	In(s3)	Log_state(s3) Log_trans(t7) In(s2)	In(s3)
O8	In(s3)	Log_state(s3) Log_trans(t8) In(final)	In(s3)
O9	In(s3)	Log_state(s3) Log_trans(t9) In(final)	In(s3)
O10	In(s3)	Log_state(s3) Log_trans(t10) In(final)	In(s3)
O11	In(s3)	Log_state(s3) Log_trans(t11) In(final)	In(s3)

El estado inicial y final del sistema dependen del conjunto de pruebas que se quiera desarrollar, ya que como se puede ver en el diagrama de estados, existen distintos finales según la secuencia de eventos / transiciones ejecutada. Un posible estado inicial y final se describe en la tabla 57.

Tabla 57. Estado inicial y final del sistema.

Estado inicial	In(Inicial)
Estado final	Log_trans(t11), In(final)

A partir del conjunto de operadores sólo es necesario aplicar un algoritmo que genere todas las posibles secuencias. Cada una de esas secuencias será un caso de prueba. A continuación, en la tabla 58, se muestra, como resultado de esta propuesta, dos posibles secuencias de ejecución a partir del estado inicial y final de la tabla 57.

Tabla 58. Secuencias generadas.

Secuencia 1:
Inicial s0 s0 s1 s1 s2 s3 final
Secuencia 2:
Inicial s0 s1 s2 s3 final

En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.3. Generación del conjunto de pruebas aplicando Requirements by Contract

El primer paso de esta propuesta consiste en ampliar el diagrama de casos de uso con un contrato para cada uno de los casos. Estos contratos se recogen en la tabla 59.

Tabla 59. Contratos para los casos de uso.

Caso de uso Entrada al sistema:
UC entrada() **Pre --**

Post UsuarioValido(u)
Caso de uso Préstamo de libros:
UC prestamo(usuario u) **Pre** UsuarioValido(usuario u) **Post** LibroPrestado(l)
Caso de uso Devolución de libros:
UC devolucion(usuario u, libro l) **Pre** UsuarioValido(usuario u) and LibroPrestado(l) **Post not** LibroPrestado(l)

Esta propuesta permite definir con total libertad predicados con significado semántico propio. A continuación, en la tabla 60, se describe la semántica de los predicados utilizados en los contratos de la tabla 59.

Tabla 60. Descripción de la semántica de los predicados.

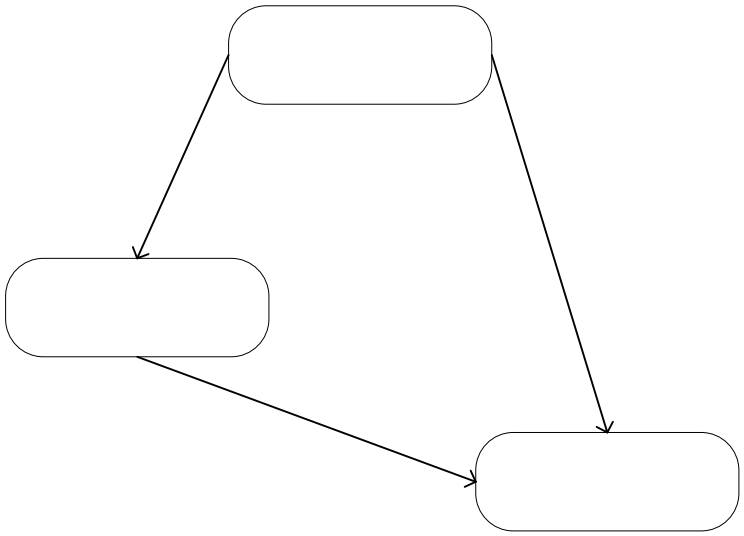
Predicado	Descripción
UsuarioValido(u)	Indica que el usuario u debe estar validado mediante la ejecución del caso de uso Entrada al sistema
LibroPrestado(l)	Indica que el libro l está en préstamo
Not LibroPrestado(l)	Indica que el libro l está disponible para préstamo

A partir de los parámetros y de las precondiciones y poscondiciones expresadas en los contratos de la tabla 59, se puede ver el orden en que deben ser ejecutados los casos de uso.

En concreto, la tabla 59 expresa que el caso de uso Entrada al sistema debe ser ejecutado antes que los otros dos casos de uso y que el caso Préstamo de libro debe ser ejecutado antes que el caso de uso Devolución de libro.

A continuación se construye el modelo de ejecución de casos de uso. Este modelo muestra la evolución de los distintos parámetros de los casos de uso. Los estados muestran el estado del sistema en función de los predicados definidos en los contratos. Cada transición representa la ejecución de un caso de uso. El modelo de ejecución se muestra en la ilustración 23.

Ilustración 23. Modelo de ejecución.



A continuación se selecciona el criterio de cobertura. Para este caso práctico se ha seleccionado el criterio de todos los estados y transiciones. Todos los estados y todas las transiciones del modelo de la ilustración 23 deben estar contenidos en, al menos, una prueba.

Para generar el conjunto de pruebas, se ha utilizado la herramienta gratuita disponible en [RBCTool]. La descripción del modelo de la ilustración 17 para poder ser utilizado en dicha herramienta se recoge en la tabla 61.

Tabla 61. Modelo del sistema.

```
# Entities of the system
{
        u1 : USUARIO
        l1, l2   : LIBRO
}
# Description of the initial state
{
   disponible(l1)
   disponible(l2)
}
# Use cases description
#use case EntradaSistema
UC EntradaSistema(U:USUARIO)
pre
post uvalido(U)
```

```
#use case Borrow
UC Borrow(U:USUARIO; L:LIBRO)
pre uvalido(U) and disponible(L)
post prestamo(L) and not disponible(L)
#use case Return
UC Return(U:USUARIO; L:LIBRO)
pre uvalido(U) and prestamo(L)
post not prestamo(L) and disponible(L)
```

A continuación, en las ilustraciones 24 y 25, se muestran algunas capturas de la herramienta tomadas mientras se ejecutaba el modelo del sistema de la ilustración 22.

Ilustración 24. Carga de datos al sistema.

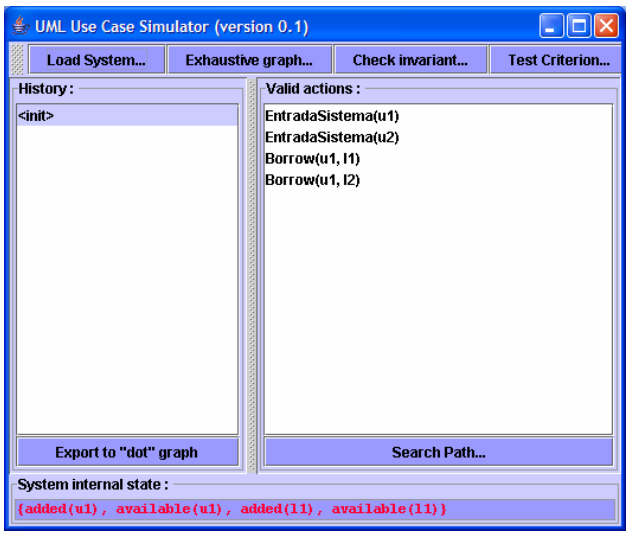
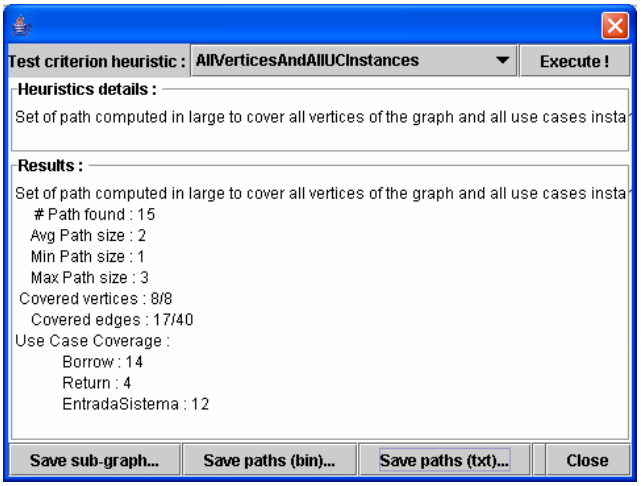


Ilustración 25. Estadísticas de la generación de pruebas.



El resultado de aplicar esta herramienta sobre el modelo de ejecución de la Ilustración 23 se muestra en la tabla 62. Se ha variado el criterio de cobertura para comparar las distintas secuencias de casos de uso generadas a partir de cada criterio.

Tabla 62. Secuencias generadas.

Criterio de todos los bordes.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Borrow(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l2)]
Criterio de todos los vértices.
[EntradaSistema(u1), Borrow(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2)]
Criterio de todos los casos de uso instanciados.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
Ce qu'il faut couvrir : [EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1), Return(u1, l2)] : 5
Criterio de todos los vértices y casos de uso instanciados.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2)]
Ce qu'il faut couvrir : [EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1), Return(u1, l2)] : 5

En el criterio de todos los casos de uso instanciados y de todos los vértices y casos de uso instanciados, la herramienta lanza un error y el proceso de generación se detiene.

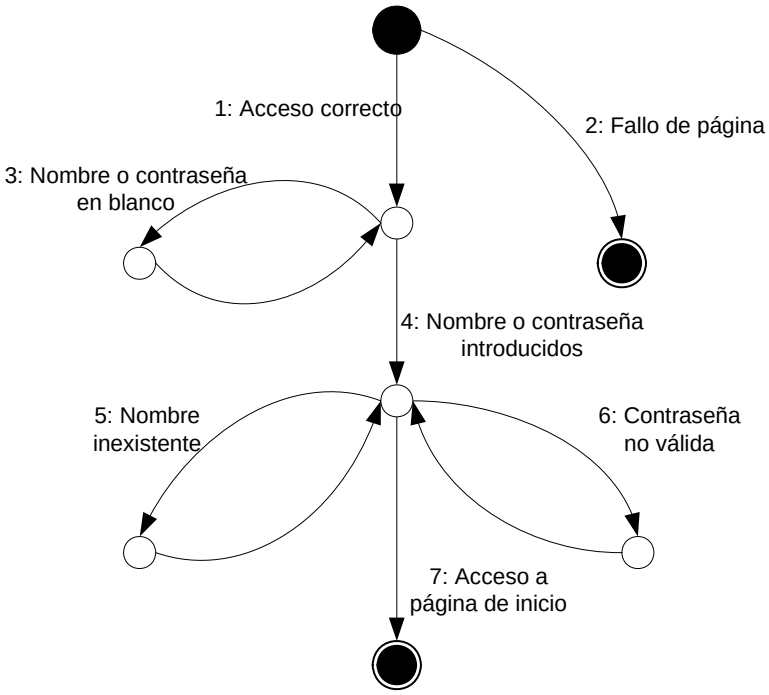
En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.4. Generación del conjunto de pruebas aplicando Use Case Path Analysis.

Como se ha visto en sección 3, esta propuesta sólo permite generar pruebas que verifiquen el comportamiento de cada caso de uso de manera aislada, sin tener en cuenta dependencias con otros casos de uso. Por este motivo, sólo vamos a aplicar esta propuesta al primero de los casos de uso.

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 26 en el punto 2.5. Este diagrama incluye, además, un número único para cada rama, lo que facilitará la tarea a la hora de identificar los caminos.

Ilustración 26. 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 63 se muestran todos los caminos identificados a partir del diagrama de flujo anterior.

Tabla 63. Caminos posibles de ejecución.

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.

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 cada camino según la fórmula: Valoración = Frecuencia + Factor crítico. A continuación, en la tabla 64, se muestra las puntuaciones de cada atributo y la valoración de cada camino identificado.

Tabla 64. Atributos y factor para cada camino.

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

A continuación se ordenan los caminos en función de su valoración. Los caminos ordenador de mayor a menor valoración se muestran en la tabla 65.

Tabla 65. Caminos ordenador según su valoración.

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

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 crítico 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 generarse 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 66.

Tabla 66. Caminos seleccionados para derivarlos en casos de prueba.

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.

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 generar 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 67 mostramos un conjunto de datos de prueba para el camino Id 5. En cursiva se muestran los valores válidos.

Tabla 67. Datos de prueba.

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.

En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

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

Por los mismos motivos que la propuesta anterior, sólo vamos a aplicar esta propuesta al primero de los casos de uso.

La parte más importante de un caso de uso para la generación automática de casos de prueba es el camino de ejecución. Este camino se divide en el camino principal o secuencia normal y en 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 en la tabla 68.

Tabla 68. Todos los posibles escenarios de uso para el caso de uso en estudio.

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

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 69.

Tabla 69. Matriz de casos de prueba.

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.

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 70.

Tabla 70. Matriz de casos de prueba con sus valores de prueba.

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	“” 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.

Repitiendo este proceso por cada caso de uso obtendremos un catálogo con las pruebas del sistema necesarias para garantizar la calidad del sistema desarrollado.

En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.6. Generación del conjunto de pruebas aplicando PLUTO y Category-Partition Method.

El primer paso de esta propuesta es la descomposición de la especificación funcional del sistema en unidades funcionales. PLUTO propone tomar como unidades funcionales los propios casos de uso, por lo que, en este ejemplo, cada caso de uso de la ilustración 14 será una unidad funcional.

En segundo lugar se identifican los parámetros y condiciones del entorno de ejecución que afectan a la ejecución de cada unidad funcional. No existe una guía precisa para identificar estos elementos, por lo que es necesario guiarse por las características propias del problema y la intuición y experiencia de los ingenieros de prueba. Los parámetros y condiciones relevantes del entorno para cada unidad funcional del caso práctico se describen en la tabla 71.

Tabla 71. Parámetros y condiciones del sistema.

Predicado	**Parámetros y condiciones**

Entrada al sistema	Estado del servidor Nombre del usuario Clave del usuario
Préstamo de libro	Usuario Identificación del libro Estado del libro
Devolución de libro	Usuario Identificación del libro Estado del libro

Cada uno de los parámetros y condiciones de la tabla 70 va a tener su propia categoría. Esto significa que cada uno de los parámetros y condiciones del sistema es una característica relevante que debe ser probada sobre el sistema modificando su valor.

A continuación se parte cada categoría en un conjunto de elecciones. Las elecciones nos indican los posibles rangos de valores de cada categoría. La tabla 72 muestra las elecciones de cada una de las categorías.

Tabla 72. Listado de posibles elecciones.

Categorías	**Elecciones**
Estado del servidor	Activo Inactivo
Nombre del usuario	Vacío Nombre válido Nombre inválido
Clave del usuario	Vacío Clave válida Clave inválida
Usuario	Validado No validado
Identificador de libro	Vacío Identificador válido Identificador inválido
Estado del libro	Disponible No disponible.

A continuación se identifican las restricciones de cada elección. Una restricción es una condición que se tiene que cumplir para que una categoría pueda tener un valor. Estas restricciones permiten que el valor de una categoría dependa del valor de otras

categorías. Para expresar las restricciones, CP Method propone utilizar sentencias if. Las elecciones que necesitan restricciones se muestran en la tabla 73.

Tabla 73. Restricciones de las elecciones.

Elección	Restricción
Usuario Validado	**If** nombre valido **and** clave válida

Las especificaciones de prueba consisten en la traducción a lenguaje TSL de toda la información anterior. Esto se debe hacer con una herramienta que automatice el proceso. Ni PLUTO ni el documento sobre CP Method describen cómo generar las especificaciones de prueba ni ofrecen ninguna referencia a herramientas que permitan realizar ese proceso.

En el punto 4.13 se muestra un caso de prueba tomado del ejemplo de PLUTO y adaptado a este ejemplo.

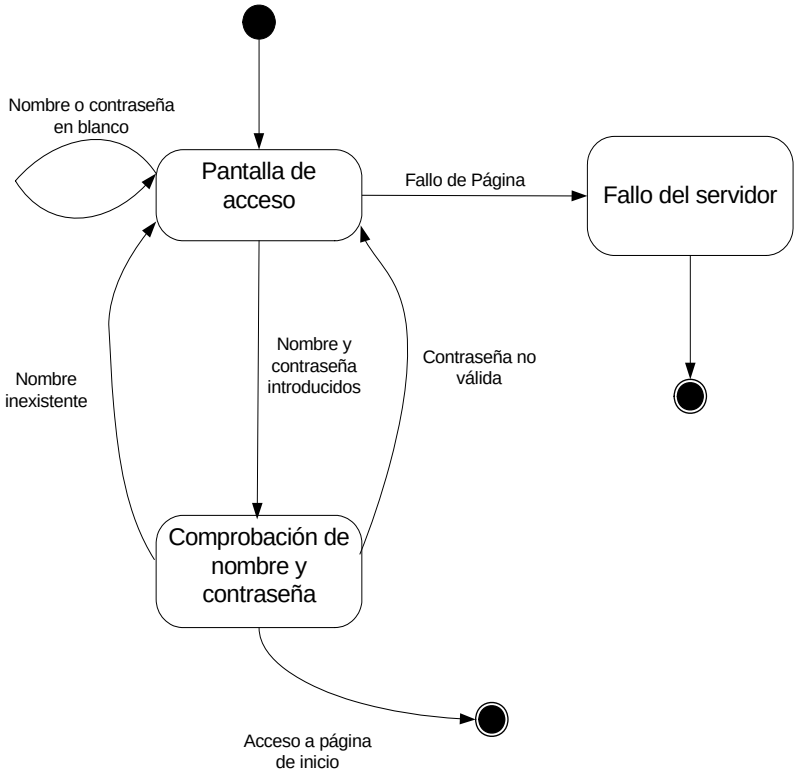
4.7. Generación del conjunto de pruebas aplicando SCENT.

En este ejemplo vamos a centrarnos solamente en el proceso de generación 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 primer caso de uso descrito en el punto 4.1, el cuál es lo suficientemente completo para poder aplicar directamente la generación de casos de prueba.

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 en la ilustración 27.

A continuación, aplicaremos sólo el segundo bloque de la propuesta y solo el primer y segundo punto de la misma, los únicos que son obligatorios.

Ilustración 27 . Diagrama de estados correspondiente al caso de uso Entrada al sistema.



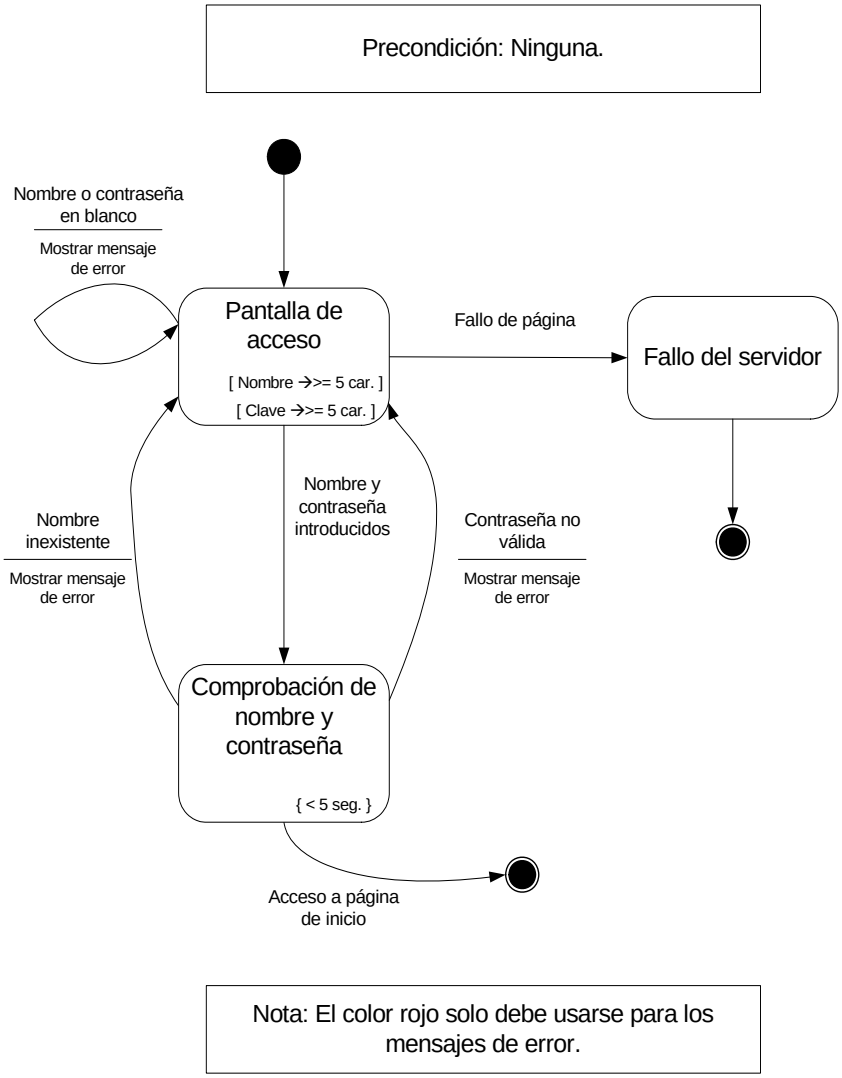
Este diagrama de estados es necesario extenderlo con precondiciones, poscondiciones, datos de entrada, los datos de salida esperados y requisitos no funcionales, tales como requisitos de ejecución.

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, 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 28.

Ilustración 28. Diagrama de estados con información extendida.



En SCENT los casos de prueba para pruebas de sistema son generados 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 74). 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 determinada por las precondiciones.

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 de la ilustración 28 se muestra a continuación en la tabla 74. Por ejemplo, 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.

Tabla 74. Casos de prueba del caso de uso.

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.

Técnicas de prueba de dominio y pruebas de flujos de datos pueden aplicarse para derivar casos de prueba adicionales. Esta propuesta no indica como hacerlo.

En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

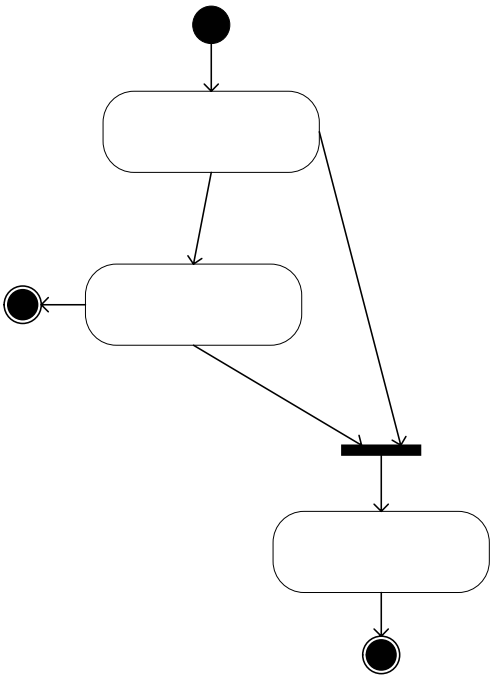
4.8. Generación del conjunto de pruebas aplicando TOTEM.

Como ya se ha comentado en el punto 3.7, esta propuesta no describe todos los puntos de su proceso de generación de pruebas. Por ese motivo nos limitaremos a aplicar sólo los puntos descritos en la documentación.

El primer punto es la derivación de secuencias de dependencias de casos de uso. Tal y como propone la especificación UML, las dependencias entre casos de uso se expresan mediante un diagrama de actividades. TOTEM propone añadir a este diagrama parámetros para los distintos casos de uso.

TOTEM propone construir un diagrama de actividades por cada actor del sistema. En nuestro ejemplo contamos con un único usuario por lo que solo es necesario construir un diagrama de actividades. El diagrama de actividades con los casos de uso de este ejemplo y sus parámetros se muestra en la ilustración 29.

Ilustración 29. Diagrama de actividades de los casos de uso.



A partir de este diagrama se generan secuencias válidas de casos de uso. Una secuencia válida es la ejecución de un conjunto de casos de uso. Estas secuencias se pueden generar de manera automática a partir del diagrama de actividades. Se deben generar tantas secuencias como posibles combinaciones existan. Un conjunto de combinaciones a partir del diagrama de la ilustración 29 se muestra en la tabla 75.

Tabla 75. Secuencias de ejecución de casos de uso.

Número	Secuencia
1	EntradaSistema(u1).PrestamoLibro(u1, l1)
2	EntradaSistema(u1).PrestamoLibro(u1, l1).DevolucionLibro(u1, l1)
3	EntradaSistema(u2).PrestamoLibro(u2, l2)
4	EntradaSistema(u2).PrestamoLibro(u2, l2).DevolucionLibro(u2, l2)

Además, las secuencias deben repetirse con parámetros distintos. El número de repeticiones de cada secuencia queda a la discreción de los ingenieros de prueba. En este caso, por simplicidad, hemos decidido repetir cada secuencia una segunda vez. Estas repeticiones también se han incluido en la tabla 75.

A continuación, las secuencias de la tabla 75 deben combinarse entre sí. Algunas combinaciones posibles se muestran en la tabla 76.

Tabla 76. Combinaciones de secuencias de ejecución de casos de uso.

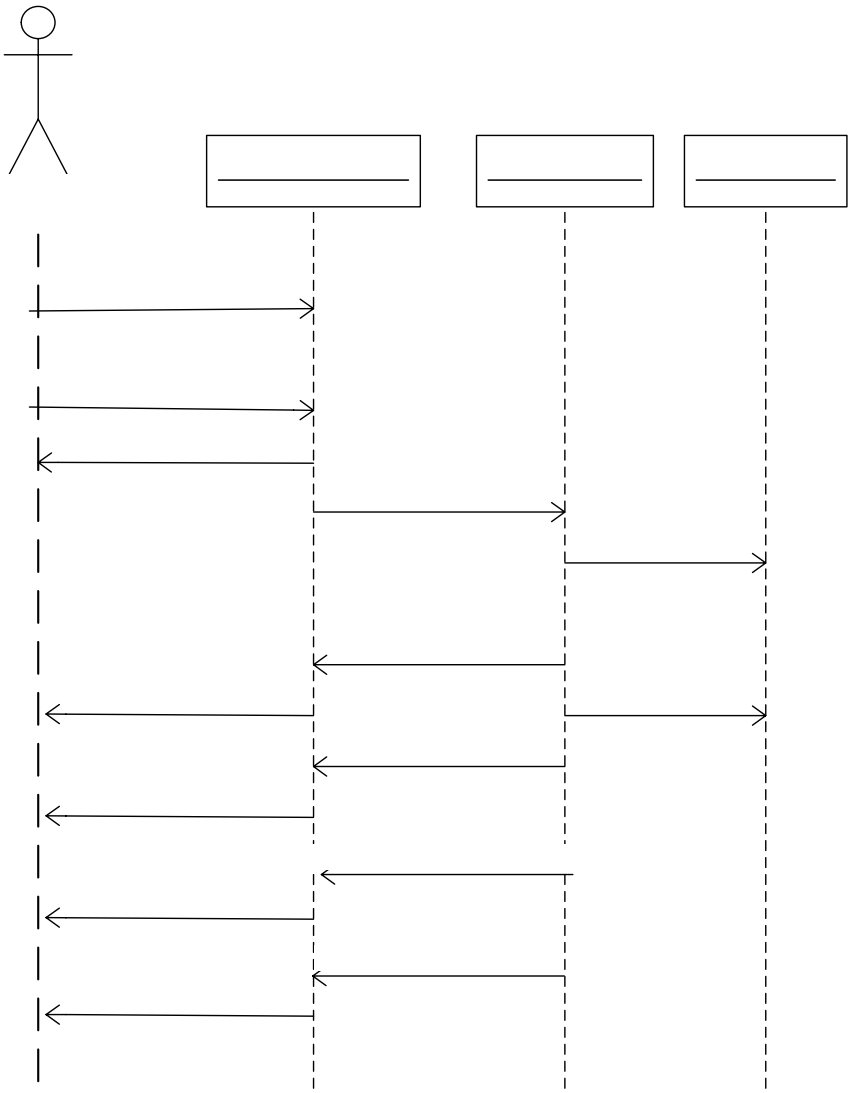
Número	Secuencia
1	EntradaSistema(u1). EntradaSistema(u2).PrestamoLibro(u2, l2).PrestamoLibro(u1, l1)
2	EntradaSistema(u1). EntradaSistema(u2).PrestamoLibro(u2, l2). PrestamoLibro(u1, l1).DevolucionLibro(u1, l1). DevolucionLibro(u2, l2)

Todas las secuencias generadas deberán ser probadas en la implementación del sistema. Según la propuesta todo este proceso puede automatizarse, pero no se ofrece ninguna referencia a ninguna herramienta que permita hacerlo.

A continuación, como una tarea independiente de los resultados anteriores, se generan pruebas para cada caso de uso en particular. Como ejemplo se van a generar pruebas para el caso de uso Entrada al sistema.

El punto de partida es un diagrama de secuencias con todas las posibles interacciones entre las clases participantes. El diagrama de secuencias correspondiente a dicho caso de uso ha sido realizando siguiendo la notación gráfica de la propuesta y se muestra en la ilustración 30.

Ilustración 30. Diagrama de secuencia del caso de uso Entrada al sistema.



A continuación se representan las secuencias del diagrama de la ilustración 30 en forma de expresiones regulares. El alfabeto de dichas expresiones regulares son los métodos públicos de las clases participantes en el diagrama de secuencias. Por comodidad, todos los métodos públicos se recogen en la tabla 77.

Tabla 77. Métodos públicos del diagrama de secuencias.

Clase	**Secuencia**
FormularioAcceso	Entrada(nombre, clave)
FormularioAcceso	nombreOClaveEnBlanco()
ControlAcceso	Comprobar(nombre, clave)
ControlAcceso	Existe(nombre / clave)
ControlAcceso	NoExiste(nombre / clave)

ControlAcceso	NoCoinciden(nombre, clave)
FormularioAcceso	MensajeError()
ControlAcceso	Coinciden(nombre, clave)
FormularioAcceso	MensajeAccesoPermitido()

A continuación, en la tabla 78, se muestran algunas posibles expresiones regulares, obtenidas a partir del diagrama de secuencias de la ilustración 30, y expresadas según la notación propuesta en la documentación.

Tabla 78. Expresiones regulares.

Expresiones regulares
Entrada al sistema -> (entrada $_{FormularioAcceso}$. nombreOClaveEnBlanco$_{FormularioAcceso}$)*
Entrada al sistema -> entrada $_{FormularioAcceso}$. comprobar $_{ControlAcceso}$. existe $_{ControlAcceso}$. existe $_{ControlAcceso}$. coincide $_{ControlAcceso}$. mensajeAccesoPermitido $_{FormularioAcceso}$

A continuación, se fusionan las expresiones regulares en una suma de productos. El resultado de fusionar las expresiones regulares de la tabla 78 se muestra en la tabla 79.

Tabla 79. Expresiones regulares fusionadas.

Expresiones regulares
Entrada al sistema -> (entrada $_{FormularioAcceso}$. nombreOClaveEnBlanco$_{FormularioAcceso}$)* + entrada $_{FormularioAcceso}$. comprobar $_{ControlAcceso}$. existe $_{ControlAcceso}$. existe $_{ControlAcceso}$. Coincide $_{ControlAcceso}$. mensajeAccesoPermitido $_{FormularioAcceso}$

A continuación se identifican las condiciones necesarias para poder generar productos a partir de las expresiones regulares fusionadas obtenidas. Estas condiciones se expresan mediante OCL [OCL03]. En la tabla 80 se muestran las condiciones que deben cumplirse para que las dos expresiones regulares de la tabla 78 puedan ejecutarse.

Tabla 80. Condiciones.

	Expresiones regulares
A	Self.FormularioAcceso.nombre -> exists(n: nombre | n="") and exists(c: clave | c="")

B	Self.FormularioAcceso.nombre -> exists(n: nombre | n<>"") and exists(c: clave | c<>"") and (self.ControlAcceso.coincide(n, c) == true)

Teniendo identificadas las condiciones bajo las que cada término puede ejecutarse y, por tanto, probarse, es necesario identificar las secuencias de operaciones precisas que serán ejecutadas para cada término. Esto se consigue expandiendo los operadores * y +. El número de repeticiones se deja a criterio de los ingenieros de prueba. En la tabla 81 se muestran varias secuencias de operaciones precisas generadas a partir de la variación del primer término de la expresión regular de la tabla 79.

Tabla 81. Expresiones regulares precisas.

Repeticiones	Expresiones regulares
0	Entrada al sistema -> entrada $_{FormularioAcceso}$. comprobar $_{ControlAcceso}$. existe $_{ControlAcceso}$. existe $_{ControlAcceso}$. coincide $_{ControlAcceso}$. mensajeAccesoPermitido $_{FormularioAcceso}$
1	Entrada al sistema -> entrada $_{FormularioAcceso}$. nombreOClaveEnBlanco$_{FormularioAcceso}$ .entrada $_{FormularioAcceso}$. comprobar $_{ControlAcceso}$. existe $_{ControlAcceso}$. existe $_{ControlAcceso}$. coincide $_{ControlAcceso}$. mensajeAccesoPermitido $_{FormularioAcceso}$
2	Entrada al sistema -> entrada $_{FormularioAcceso}$. nombreOClaveEnBlanco$_{FormularioAcceso}$. entrada $_{FormularioAcceso}$. nombreOClaveEnBlanco$_{FormularioAcceso}$ .entrada $_{FormularioAcceso}$. comprobar $_{ControlAcceso}$. existe $_{ControlAcceso}$. existe $_{ControlAcceso}$. coincide $_{ControlAcceso}$. mensajeAccesoPermitido $_{FormularioAcceso}$

A continuación se identifican los oracles de prueba [Binder00]. Este proceso se desarrolla de manera automática, transformando las condiciones expresadas en OCL identificadas con anterioridad. Esta propuesta solo da una visión muy general de cómo realizar esta tarea, por lo que no la tendremos en cuenta en este caso práctico.

Por último se construyen tablas de decisión. En la tabla 82 se muestra la tabla de decisiones para las expresiones regulares de la tabla 81.

Tabla 82. Tabla de decisión.

Variantes	Condiciones		Acciones		
	A	B	Mensaje I	Mensaje II	Cambio estado
1	Sí	No	Sí	No	No
2	Sí	Sí	Sí	Sí	Sí
3	Sí	Sí	Sí	Sí	Sí

Los posibles mensajes al actor usuario se describen en la tabla 83.

Tabla 83. Mensajes al actor usuario.

	Mensajes
I	nombreOClaveEnBlanco
II	mensajeAccesoPermitido

Estas tablas de decisión involucran a casos de uso de manera aislada. Según la propuesta, a continuación sería necesario construir tablas de decisiones que involucraran a las secuencias de casos de uso generadas al principio de este punto. Sin embargo TOTEM solo describe como hacer esto de una manera muy superficial y no muestra ningún ejemplo, por lo que no se tendrá en cuenta en este caso práctico.

En el punto 4.12 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.9. Generación del conjunto de pruebas aplicando Extended Use Cases Test Design Pattern.

Como ya se ha comentado en el punto 3.9, esta propuesta sólo permite obtener casos de prueba a partir de casos de uso de manera aislada. Por este motivo sólo se va a aplicar al caso de uso Entrada al sistema.

El primer paso es la identificación de las variables operacionales. Como se ha comentado en la descripción de esta propuesta, las variables operacionales son los factores que varían entre distintas ejecuciones de un caso de uso. Las variables operacionales identificadas para el caso de uso 1 se enumeran en la tabla 84.

Tabla 84. Variables operacionales del caso de uso Entrada al sistema.

Número	Variables operacionales
1	Conexión al sistema.
2	Nombre del usuario.
3	Clave del usuario.

A continuación se identifica el dominio de cada una de las variables operacionales. Estos dominios se muestran en la tabla 85.

Tabla 85. Dominios de las variables operacionales.

Número	Variables operacionales	Dominio
1	Conexión al sistema.	Conectado / Desconectado
2	Nombre del usuario.	Secuencia de caracteres o cadena vacía
3	Clave del usuario.	Secuencia de caracteres o cadena vacía

A continuación se especifican las relaciones operacionales. Estas relaciones involucran todos los posibles valores de todas las variables operacionales. Las relaciones operacionales de este ejemplo de muestran en la tabla 86.

Tabla 86. Relaciones operacionales

Núm.	Variables operacionales			Resultado esperado
	Conexión al sistema.	Nombre de usuario	Clave del usuario.	
1	Desconectado	--	--	Mensaje de error
2	Conectado	Cadena vacía	--	Solicitud de nombre y clave.
3	Conectado	--	Cadena vacía	Solicitud de nombre y clave.
4	Conectado	Nombre inválido	--	Mensaje de error
5	Conectado	Nombre válido	Clave inválida	Mensaje de error
6	Conectado	Nombre válido	Clave válida	Mensaje de error

Por último se desarrolla un caso de prueba por cada relación operacional identificada. Este punto no se describe en la propuesta.

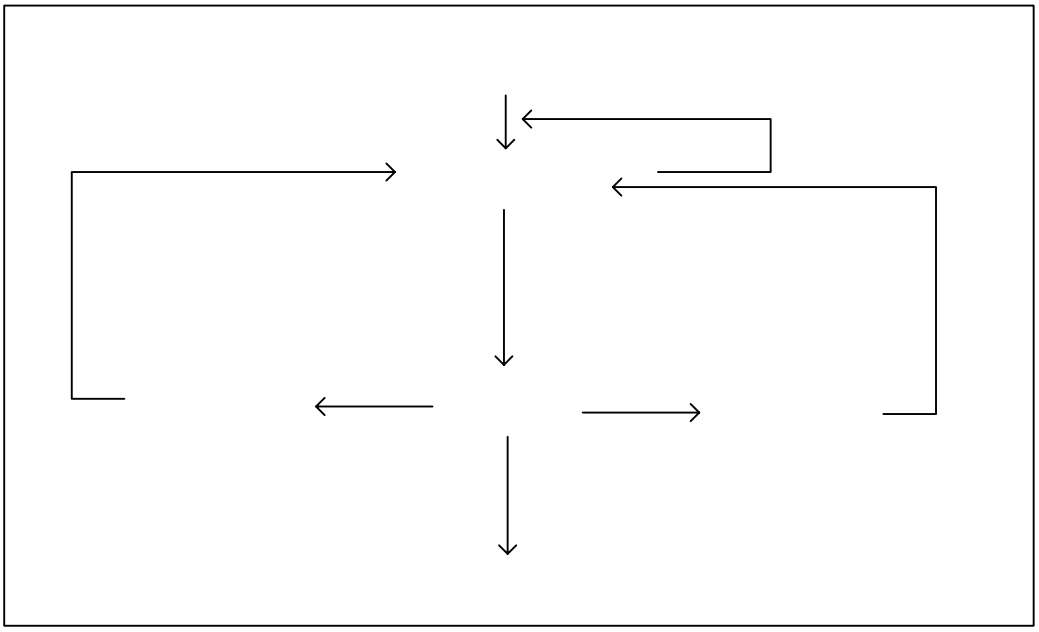
En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.10. Generación del conjunto de pruebas aplicando Software Requirements and Acceptance Testing.

Esta propuesta se va a aplicar al caso de uso Entrada al sistema.

En primer lugar se construye un árbol de escenario a partir de cada requisito. El árbol de escenario correspondiente al caso de uso Entrada al sistema se muestra en la ilustración 31.

Ilustración 31. Árbol de escenario para la entrada al sistema.



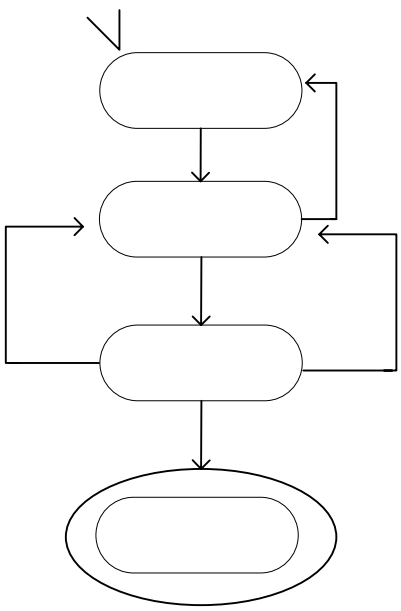
A partir de dicho árbol de escenario se extrae una gramática. La gramática correspondiente al árbol de la ilustración 31 se muestra en la tabla 87.

Tabla 87. Gramática para la entrada al sistema.

Eventos = { Acceso, Nombre o contraseña en blanco, Entrar autentificación, Nombre inexistente, Contraseña no válida, Autentificación válida }
Estados = { Mostrar formulario de autentificación, Comprobar autentificación, Mensaje de error }
Estado inicial = Usuario
Reglas = { Usuario -> Acceso (Mostrar formulario de autentificación), Mostrar formulario de autentificación -> Nombre o contraseña en blanco (Mostrar formulario de autentificación) | Entrar autentificación (Comprobar autentificación), Entrar autentificación -> Autentificación válida | Nombre inexistente (Mensaje de error) | Contraseña no valida (Mensaje de error), Mensaje de error -> Acceso (Mostrar formulario de autentificación), }

A continuación se construye una máquina de estados finitos a partir de la gramática. La propuesta incluye su propia notación para máquinas de estados finitos. La máquina de estados finitos de este ejemplo se muestra en la ilustración 32.

Ilustración 32. Máquina de estados finitos.



En este caso, la máquina de estados de la ilustración 32 es muy similar al árbol de escenarios de la ilustración 26. Este hecho se describe con más detalle en el análisis del caso práctico del punto 5.2.

Como se ha comentado en el punto 3.10, a partir de la máquina de estados se construyen matrices de caminos. Sin embargo esta propuesta no indica como hacerlo ni lo aplica en el ejemplo que incluye.

Por este motivo no es posible terminar de aplicar la propuesta.

4.11. Generación del conjunto de pruebas aplicando Use Case Derived Test Cases

Esta propuesta se va a aplicar sobre el caso de uso Entrada al sistema. La mecánica y los resultados de esta propuesta son los mismos que los resultados de aplicar la propuesta Test Cases from Use Cases. A continuación, en la tabla 88, se muestran algunos de esos mismos resultados redactaros siguiendo el ejemplo incluido en dicha propuesta

Tabla 88. Casos de prueba.

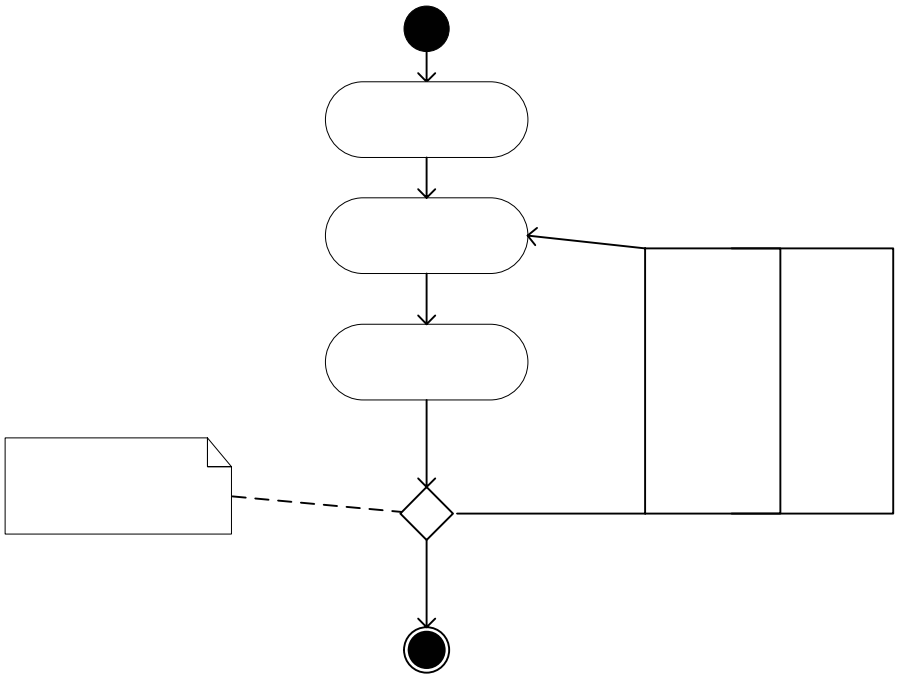
Test Condition 1: Acceso al sistema
• El usuario introduce un nombre y clave válidos
• El sistema autoriza el acceso.
Test Condition 2: Nombre en blanco
• El usuario introduce un nombre en blanco
• El sistema autoriza solicita de nuevo el nombre.

En el punto 4.13 se muestran los resultados de esta propuesta junto con los resultados de las demás propuestas.

4.12. Generación del conjunto de pruebas aplicando A Model-based Approach to Improve System Testing of Interactive Applications

En el primer paso de esta propuesta se construye un diagrama de actividades del sistema a partir de un caso de uso. El diagrama de actividades resumido correspondiente al caso de uso Entrada al sistema se muestra en la ilustración 33.

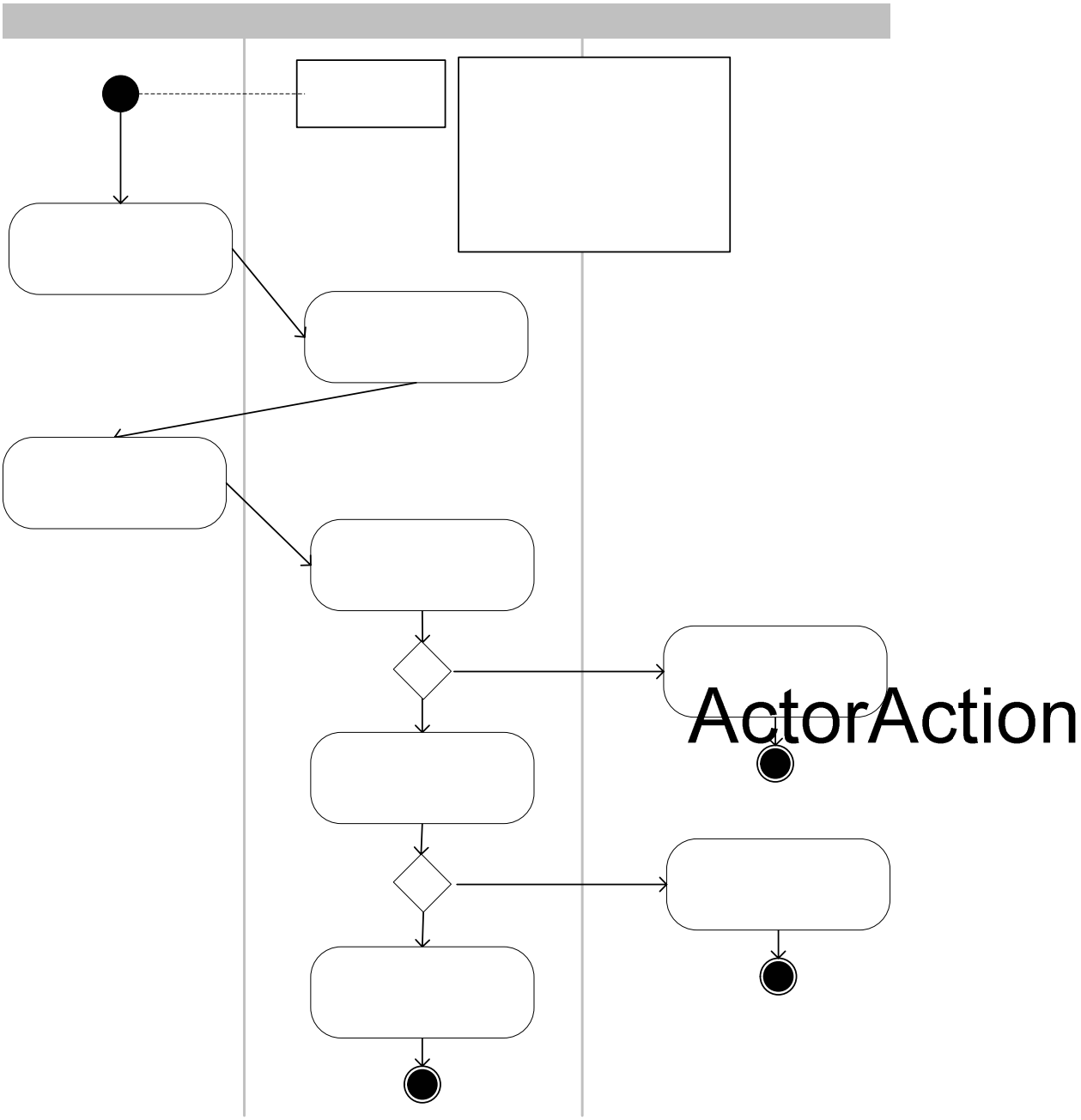
Ilustración 33. Diagrama de actividades de acceso al sistema.



A continuación completamos este diagrama de actividades con los estereotipos <<UserAction>>, <<SystemResponse>> o <<Include>>. Como solo tenemos un único

diagrama de estados, el estereotipo << Include >> no va a ser utilizado. El diagrama de actividades detallado y anotado con estereotipos se muestra en la ilustración 34.

Ilustración 34. Diagrama de actividades anotado con estereotipos.



A continuación se aplica el método de particiones en categorías para obtener un modelo del sistema en TSL. Esto es exactamente lo mismo que lo explicado para la propuesta PLUTO y presenta los mismos problemas que en el punto 4.5.

Tampoco hemos podido conseguir ninguna versión de la herramienta para obtener casos de prueba ejecutables.

Por este motivo no es posible terminar de aplicar la propuesta.

4.13. Resumen de los resultados obtenidos.

En este punto se resumen los resultados obtenidos por las propuestas que se han podido aplicar en su totalidad. Con este resumen, se puede apreciar de manera sencilla las diferencias entre los resultados obtenidos aplicando cada propuesta descrita en la sección 3 al mismo caso práctico.

4.13.1 Automated Test Case Generation from Dynamic Models.

A partir de esta propuesta se ha obtenido un conjunto de operadores que describen el comportamiento del sistema. Aplicando estos operadores a una herramienta de planificación se obtiene un conjunto de posibles secuencias de ejecución que deberán ser probadas en el sistema.

Tabla 89. Secuencias generadas.

Secuencia 1:
Inicial s0 s0 s1 s1 s2 s3 final
Secuencia 2:
Inicial s0 s1 s2 s3 final

4.13.2. Requirements by Contract

A partir de esta propuesta se ha obtenido un modelo de ejecución del sistema y una serie de secuencias de ejecución válidas de casos de uso, que deberán ser implementadas y probadas sobre el sistema bajo prueba.

Tabla 90. Secuencias generadas.

Criterio de todos los bordes.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Borrow(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l2)]
Criterio de todos los vértices.
[EntradaSistema(u1), Borrow(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2)]
Criterio de todos los casos de uso instanciados.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
Ce qu'il faut couvrir : [EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1), Return(u1, l2)] : 5
Criterio de todos los vértices y casos de uso instanciados.
[EntradaSistema(u1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l1), Return(u1, l1)]
[EntradaSistema(u1), Borrow(u1, l2), EntradaSistema(u1)]
[EntradaSistema(u1), Borrow(u1, l2), Return(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l2)]
[EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2)]
Ce qu'il faut couvrir : [EntradaSistema(u1), Borrow(u1, l1), Borrow(u1, l2), Return(u1, l1), Return(u1, l2)] : 5

4.13.3. Use Cases Using Path Analysis Technique

El resultado de esta propuesta es una tabla donde se describen las distintas secuencias de interacciones que serán convertidas en casos de prueba ordenadas por prioridades.

Tabla 91. Caminos seleccionados para derivarlos en casos de prueba.

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.

4.13.4. Test cases form Use Cases.

El resultado de esta propuesta es una tabla con todos los casos de prueba que deberán ser implementados, sus valores de prueba y resultados esperados.

Tabla 92. Matriz de casos de prueba con sus valores de prueba.

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.

4.13.5. PLUTO y Category Partition Method

El resultado de esta propuesta son casos de prueba expresados en el lenguaje TSL. En la tabla 93 se muestra un caso de prueba adaptado al caso práctico descrito en el punto 4.1.

Tabla 93. Caso de prueba.

Tag Nombre Uusuario=Nombre inválido Ti: Scenarios: ext: Nombre inválido.

4.13.6. SCENario-Based Validation and Test of Software (SCENT)

SCENT genera una tabla con todos los casos de prueba, la salida esperada y el estado del sistema.

Tabla 94. Casos de prueba del caso de uso.

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	Fallo del servidor	Mensaje de fallo de página.

	servidor		

4.13.7. TOTEM

Esta propuesta genera varios resultados. En primer lugar se ha obtenido una serie de secuencias de ejecución válidas de casos de uso. En segundo lugar se ha obtenido una tabla de decisiones con cada caso de uso con las condiciones iniciales y los resultados esperados.

Tabla 95. Combinaciones de secuencias de ejecución de casos de uso.

Número	Secuencia
1	EntradaSistema(u1). EntradaSistema(u2).PrestamoLibro(u2, l2).PrestamoLibro(u1, l1)
2	EntradaSistema(u1). EntradaSistema(u2).PrestamoLibro(u2, l2). PrestamoLibro(u1, l1).DevolucionLibro(u1, l1). DevolucionLibro(u2, l2)

Tabla 96. Expresiones regulares precisas.

Repeticiones	Expresiones regulares
0	Entrada al sistema -> entrada $_{\text{FormularioAcceso}}$. comprobar $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. coincide $_{\text{ControlAcceso}}$. mensajeAccesoPermitido $_{\text{FormularioAcceso}}$
1	Entrada al sistema -> entrada $_{\text{FormularioAcceso}}$. nombreOClaveEnBlanco$_{\text{FormularioAcceso}}$ .entrada $_{\text{FormularioAcceso}}$. comprobar $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. coincide $_{\text{ControlAcceso}}$. mensajeAccesoPermitido $_{\text{FormularioAcceso}}$
2	Entrada al sistema -> entrada $_{\text{FormularioAcceso}}$. nombreOClaveEnBlanco$_{\text{FormularioAcceso}}$. entrada $_{\text{FormularioAcceso}}$. nombreOClaveEnBlanco$_{\text{FormularioAcceso}}$ .entrada $_{\text{FormularioAcceso}}$. comprobar $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. existe $_{\text{ControlAcceso}}$. coincide $_{\text{ControlAcceso}}$. mensajeAccesoPermitido $_{\text{FormularioAcceso}}$

Tabla 97. Tabla de decisión.

Variantes	Condiciones		Acciones		
	A	B	Mensaje I	Mensaje II	Cambio estado
1	Sí	No	Sí	No	No
2	Sí	Sí	Sí	Sí	Sí

3	Sí	Sí	Sí	Sí	Sí

Tabla 98. Mensajes al actor usuario.

	Mensajes
I	nombreOClaveEnBlanco
II	mensajeAccesoPermitido

4.13.8. Extended Use Case Test Design Pattern

El resultado de esta propuesta es una tabla con las relaciones operacionales que serán implementadas en casos de prueba.

Tabla 99. Relaciones operacionales

Núm.	**Variables operacionales**			**Resultado esperado**
	Conexión al sistema	**Nombre de usuario**	**Clave del usuario**	
1	Desconectado	--	--	Mensaje de error
2	Conectado	Cadena vacía	--	Solicitud de nombre y clave.
3	Conectado	--	Cadena vacía	Solicitud de nombre y clave.
4	Conectado	Nombre inválido	--	Mensaje de error
5	Conectado	Nombre válido	Clave inválida	Mensaje de error
6	Conectado	Nombre válido	Clave válida	Mensaje de error

4.13.9. Use Case Derived Test Cases.

Los resultados de esta propuesta son los mismos que los resultados del punto 4.13.4.

Tabla 100. Casos de prueba.

Test Condition 1: Acceso al sistema • El usuario introduce un nombre y clave válidos • El sistema autoriza el acceso. Test Condition 2: Nombre en blanco • El usuario introduce un nombre en blanco • El sistema autoriza solicita de nuevo el nombre.

Un análisis de los resultados obtenidos por cada propuesta en el caso práctico se realiza en el punto 5.2.

5. Análisis de la situación actual.

El análisis de las propuestas descritas en este trabajo se divide en dos apartados. En el primero se analizan las propuestas a partir de la información recogida en la sección 3: la descripción de cada propuesta, la documentación disponible y los mensajes que hemos podido intercambiar con algunos de sus autores. En el segundo apartado se analizan las propuestas en función del caso práctico presentado en la sección 4. Las conclusiones de este análisis se recogen en la siguiente sección.

5.1. Comparativa de propuestas.

Para obtener una visión global de lo que ofrece cada propuesta se han establecido diecinueve características (tabla 102) y se han evaluado éstas para cada una de las propuestas de la sección 3. Estas características han sido elegidas de tal manera que describan todas las cualidades de una propuesta y faciliten la comparación con las demás propuestas. Estudiando las columnas de la tabla 102 es posible conocer rápidamente todo lo relevante de una propuesta. Estudiando las filas de la tabla 102 es posible conocer rápidamente los diferentes tratamientos dados a una característica por parte de todas las propuestas estudiadas.

Los siguientes párrafos describen cada una de las características de la tabla 102.

Por comodidad y para poder incluir toda la tabla en una única página, se han asignado siglas a cada una de las propuestas. La tabla 101 recoge las siglas asignadas a cada propuesta en la tabla comparativa (tabla 102).

Tabla 101. Siglas asignadas a cada una de las propuestas.

Propuesta	Referencia
Automated Test Case Generation from Dynamic Models.	ATCGDM
Requirements by Contract	RBC
Testing From Use Cases Using Path Analysis Technique	UCPA
Test cases form Use Cases	TCUC
PLUTO y Category Partition Method	PLUTO
SCENario-Based Validation and Test of Software	SCENT
TOTEM	TOTEM

Requirement Base Testing	RBT
Extended Use Case Test Desgin Pattern	EUC
Software Requirements and Acceptance Testing	AT
Use Case Derived Test Cases	UCDTC
A Model-based Approach to Improve System Testing of Interactive Applications	TDEUML
RETNA	RETNA

A continuación, en los siguientes párrafos, se describe cada una de las características evaluadas en la tabla 102.

Origen: indica cuáles son los artefactos necesarios para poder aplicar cada una de las propuestas. En los casos en que la propuesta contempla también la elicitación de requisitos se indica con el término necesidades. *Dominio: Set{Necesidades, Lenguaje natural, Casos de uso, Varios}*

Nueva notación: indica si la propuesta define una nueva notación gráfica. Por ejemplo, SCENT introduce sus propios diagramas de dependencias entre escenarios y UCPA sus propios diagramas de flujo. RBC presenta su propia notación para extender diagramas de casos de uso, la cuál permite añadir precondiciones, poscondiciones e invariantes a los casos de uso pero no sigue las reglas de extensión definidas en UML [UML03]. AT presenta una notación de árboles. *Dominio: Boolean.*

Notación gráfica: indica si en algún paso de la propuesta se utilizan diagramas gráficos. Si no se utiliza notación gráfica toda la propuesta se desarrolla mediante lenguaje natural, o plantillas de texto. *Dominio: Boolean.*

Casos prácticos: indica si la propuesta ofrece referencias a proyectos reales donde haya sido aplicada. Solo dos propuestas incluyen referencias a proyectos reales donde se han aplicado. *Dominio: Boolean.*

Uso de estándares: indica hasta qué punto cada propuesta utiliza diagramas y notaciones estándar o ampliamente aceptados, como UML [UML03]. *Dominio: Boolean.*

Herramientas de soporte: indica si, actualmente, existen herramientas que soporten la propuesta. *Dominio: Boolean.*

Ejemplos de aplicación: indica si una propuesta incluye ejemplos, además de casos prácticos. Todas las propuestas analizadas excepto RBT incluyen, al menos, un ejemplo de su aplicación. *Dominio: Boolean.*

Criterio de cobertura: indica el método para generar casos de prueba a partir de los casos de uso o de los modelos de comportamiento del sistema. Varios criterios significan que una propuesta incluye varios criterios distintos, como en ATCGDM, para la generación de pruebas. *Dominio: Set{Enum{Análisis de caminos, Partición de categorías, Varios, No}}.*

Valores de prueba: indica si la propuesta expone cómo seleccionar valores para los casos de prueba generados. *Dominio: Boolean.*

Optimización de casos de prueba: indica qué propuestas describen cómo seleccionar un subconjunto de los casos de prueba generados sin pérdida de cobertura de los requisitos. UCPA propone puntuar cada prueba en función de unos atributos y seleccionar sólo los que obtengan una puntuación más alta. PLUTO permite definir las particiones buscando minimizar el número de pruebas generadas. RBC ofrece distintos criterios para disminuir el número de pruebas sin comprometer su cobertura. EUC propone aplicar la norma IEEE 982.1 [Srivastava97] para calcular el grado de cobertura de los requisitos por parte del conjunto de casos de prueba generado. *Dominio: Boolean.*

Dependencias de casos de uso: indica si una propuesta puede generar pruebas que involucren más de un caso de uso o sólo puede generar pruebas a partir de casos de uso aislados. *Dominio: Boolean.*

Orden de los casos de prueba: indica si la propuesta describe el orden en que deben ejecutarse los casos de prueba generados. *Dominio: Boolean.*

Construcción del modelo: indica si la propuesta incluye la construcción de un modelo de comportamiento del sistema o aborda directamente la generación de casos de pruebas a partir de la información de los casos de uso. *Dominio: Boolean.*

Prioridad de casos de uso: indica si la propuesta es capaz de generar pruebas con mayor detalle para los requisitos o casos de uso más importantes, y generar pruebas menos detalladas para casos de usos secundarios. *Dominio: Boolean.*

Propuesta activa. Para este estudio se ha intentado contactar con los autores de todas las propuestas para conocer el estado de las mismas. Hemos preguntado a cada autor si sigue trabajando en su propuesta, si existen herramientas de soporte que se estén desarrollando, etc. Los resultados de estas consultas se recogen en la característica propuesta activa. *Dominio: Enum{Sí, no, ?}.*

Momento de comienzo: describe en qué momento la propuesta indica que puede comenzar el proceso de generación de casos de prueba de sistema. *Dominio: Enum {Elicitación de requisitos, Análisis del sistema}.*

Resultados: describe los resultados obtenidos al aplicar las distintas propuestas. Estos resultados se han mostrado en el caso práctico de la sección 4. *Dominio: Set{Enum{Pruebas en lenguaje no formal, Scripts de prueba. Modelo de prueba, Secuencia de transiciones}}.*

Calidad de la documentación: cuantifica el nivel de dificultad de aplicación de la propuesta a partir de la información disponible. Una calidad alta significa que es posible aplicar la propuesta sólo con la documentación disponible. Una calidad media significa que es posible aplicar la propuesta pero existirán lagunas o vacíos no resueltos en la documentación. Una calidad baja indica que es muy difícil aplicar la propuesta solo con la documentación disponible. *Dominio: Set{ Alta, Media, Baja}*

Pasos: Dominio: Entero.

Tabla 102. Comparativa de propuestas.

	ATCGDM	SCENT	TCUC	UCPA	PLUTO	RBC	TOTEM	RBT	EUC	AT	UCDTC	TDEUML	RETNA
Origen	Casos de uso	Necesidades	Casos de uso	Casos de uso	Casos de uso (17)	Casos de uso	Varios (13)	Casos de uso	Casos de uso	Necesidades	Casos de uso	Casos de uso	Necesidades
Nueva notación	No	Sí	No	Sí	No	Sí (11)	Sí (12)	No	No	Sí (2)	No	Sí (3)	No
Notación gráfica	Sí	Sí	No	Sí	No	Sí	Sí	Sí	No	Sí	No	Sí	No
Casos prácticos	No	Sí	No	No	No	No	No	No	No	Sí	No	No	No
Uso de estándares	Sí	No	No	No	No	Sí	Sí	Sí (8)	No	No	Sí	Sí	Sí
Herramientas de soporte	Sí (14)	No	No	No	Sí (15)	Sí	No	Sí (9)	No	No	No	Sí (9)	Sí
Ejemplos de aplicación	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí
Criterio de cobertura	Varios (6)	Análisis de caminos	Análisis de caminos	Análisis de caminos	Particiones de categorías	Varios	Varios	No (10)	Análisis de caminos	Varios	Análisis de caminos	Varios	Análisis de caminos
Valores de prueba	No	No	No	No	Sí	No	No	No	Sí	No	No	Sí	No
Optimización de pruebas	No	No	No	Sí	Sí	Sí	No	No	IEEE	No	No	No	No
Dependencias de casos de uso	Sí	Sí	No	No	Sí	Sí	Sí	No	No	Sí	No	No	No indicado
Construcción del modelo	Sí	Sí	No	Sí	No	Sí	Sí	Sí	No	Sí	No	Sí	Sí
Prioridad de casos de uso	No	No	No	Sí	Sí	No	Sí	No	No	No	No	Sí	Sí
Propuesta activa (5)	?	No	?	?	?	Sí	No	?	?	?	?	Sí	No
Momento de comienzo	Elicitación de requisitos	Elicitación de requisitos	Elicitación de requisitos	Elicitación de requisitos	Elicitación de requisitos	Elicitación de requisitos	Análisis del sistema	Elicitación de requisitos	Elicitación de requisitos (18)	Elicitación de requisitos (16)	Elicitación de requisitos	Elicitación de requisitos	Elicit.de requisitos
Resultados	Secuencias de transicionespara cada caso de uso	Pruebas en lenguaje no formal	Pruebas en lenguaje no formal	Pruebas en lenguaje no formal	Pruebas en lenguaje no formal	Secuencias de transiciones entre casos de uso	Modelo de prueba y Secuencias de transiciones	Pruebas en lenguaje no formal	Pruebas en lenguaje no formal	Modelo de prueba	Pruebas en lenguaje no formal	Scripts de prueba ejecutables	Scripts de prueba ejecutables
Calidad de la documentación	Media	Media	Media	Media	Media	Media	Alta(7)	Baja	Media	Baja	Baja	Media (4)	Baja
Pasos	7	15+3 (1)	3	5	8	4	8	12	4	6	2	3	6
Año publicación	2000	1999 / 2000	2002	2002	2004	2000 / 2001	2002	2001	1999	1997	2002	2004	2004

(1) 15 pasos para la obtención de escenarios y 3 pasos para la generación de casos de prueba.

(2) Una notación para árboles de uso y otra para máquinas de estados finitos.

(3) Define estereotipos propios para los diagramas de actividades de UML.

(4) Aunque existe la suficiente documentación y la propuesta se describe de manera detallada, es una propuesta pensada para ser aplicada con la participación de herramientas de soporte

(5) Una interrogación significa que no ha sido posible contactar con los autores de la propuesta.

(6) Todos los estados o todas las transiciones.

(7) Sin embargo es difícil de poner en práctica ya que dicha documentación no está completa.

(8) Consideramos los diagramas causa-efecto como una herramienta ampliamente difundida.

(9) No está disponible ninguna versión de evaluación ni ninguna licencia educativa.

(10) El criterio de cobertura ha sido implementado directamente en la herramienta de soporte y no se ha encontrado ninguna referencia al mismo.

(11) Diagramas de transición de casos de uso.

(12) Define una notación de texto para representar secuencias de ejecución de casos de uso.

(13) Diagramas UML de casos de uso, descripciones textuales de los casos de uso, diagramas de secuencia o colaboración por cada caso de uso, diagrama de clases con las clases del dominio de la aplicación, diccionario de datos con la descripción de cada clase, método y atributo, restricciones expresadas en OCL.

(14) Herramientas de planificación basadas en STRIP.

(15) Herramientas para generar plantillas de prueba.

(16) Después de las actividades de elicitación, verificación y validación de requisitos.

(17) Para familias de productos.

(18) Requisitos estables.

5.2. Análisis de los casos prácticos.

En el punto anterior se han comparado las características expresadas en cada una de las descripciones de las propuestas en la sección 3. Como se ha visto en la sección 4, estas propuestas se han aplicado a un mismo práctico. En este punto se realiza una exposición de los resultados de este caso práctico y se evalúa a cada una de las propuestas en acción, excepto RETNA.

5.2.1. ATCGDM.

En la aplicación de esta propuesta hemos encontrado situaciones que no se describen en la propuesta ni aparecen en el ejemplo incluido en la misma, como, por ejemplo, cuando existen varias transiciones que comienzan y terminan en el mismo estado.

Tampoco hemos encontrado una herramienta de libre descarga basada en STRIPS para poder implementar el modelo de comportamiento.

Los resultados de esta propuesta son pobres. Solo se obtiene una secuencia de estados posibles que debe ser implementada como una prueba del sistema a partir de la información de dichos estados y de los operadores aplicados. Esto requiere un esfuerzo adicional para identificar que operadores han sido aplicados en una transición concreta.

5.2.2. RBC.

La descripción de cómo generar el diagrama de comportamiento del sistema no es suficiente para ponerla en práctica. Su notación para el diagrama de transición de casos de uso es pobre respecto a los diagramas estándares de UML, como los diagramas de estados. En algunos casos, como en el caso del estado inicial cuando no se tiene aún ningún estado, la propuesta no indica qué hacer.

Tampoco queda claro a la hora de aplicar esta propuesta, cuál es la utilidad de del diagrama de comportamiento, ya que el modelo de comportamiento implementado en la aplicación de soporte puede generarse a partir del diagrama de casos de uso extendido con contratos.

El formato de los archivos que utiliza la herramienta de soporte no se detalla en la propuesta, sin embargo, mediante la información de la web y los archivos de ejemplo

que acompañan a la herramienta, se ha podido implementar el diagrama de transición de casos de uso sin mayores dificultades.

El paso del modelo de comportamiento a la descripción que utiliza la herramienta de soporte no se describe en la propuesta, por lo que lo hemos realizado de manea intuitiva a partir de los ejemplos ya disponibles.

La aplicación de soporte está muy poco elaborada. Por ejemplo, ni siquiera es capaz de indicar la línea donde hay un error léxico o sintáctico en el modelo del sistema. El número de casos de prueba varía mucho en función del criterio. En algunos criterios se han obtenido errores no documentados cuya causa no se ha podido descubrir.

5.2.3. UCPA.

UCPA, por el hecho de ser una propuesta muy simple, ha presentado varios problemas a la hora de aplicarla en un ejemplo. En primer lugar la notación de diagramas de flujos que emplea esta propuesta es sencilla si se aplica a casos de uso simples, pero muy farragosa cuando la complejidad del caso de uso aumenta. No permite incluir en los diagramas de flujo dependencias de un caso de uso respecto de otros casos de uso. Con la notación de diagramas de flujos no es posible expresar de una manera formal las condiciones que deben cumplirse para ejecutar una transición ni tampoco condiciones que dependen en exclusiva del estado del sistema y no de acciones de los usuarios.

Esta dificultad para imponer condiciones a las transiciones es un contratiempo a la hora de generar recorridos de manera automática a partir del caso de uso. Otro problema que presenta, común a todas las propuestas, es no mostrar ningún criterio de actuación claro cuando aparecen bucles en los diagramas de flujos, sobre el número de caminos a generar.

A la hora de puntuar los caminos de ejecución o recorridos del diagrama de flujo, esta propuesta tampoco ofrece un criterio claro sobre cómo puntuar cada camino. La puntuación se deja a criterios personales, basados en la experiencia o a partir de la información disponible sobre el uso del sistema.

Otro problema es que la única manera de asegurar una cobertura completa de los requisitos es seleccionar todos los caminos independientemente de su puntuación.

La propuesta tampoco da una clara indicación de cuantos caminos elegir para generara casos de prueba, ni donde establecer el límite de puntuación que discrimine los

caminos seleccionados de los no seleccionados. Este aspecto también se deja al criterio subjetivo de quien ejecute la propuesta.

5.2.4. TCUC.

Esta propuesta prácticamente no sistematiza el proceso de generación de casos de prueba. En la aplicación práctica, los tres pasos han sido realizados siguiendo el criterio propio de la persona que lo realiza. Podemos concluir que los resultados de aplicar esta propuesta son similares a los obtenidos mediante la intuición, sin aplicar ninguna propuesta.

5.2.5. PLUTO y Category Partition Method.

PLUTO muestra como adaptar el método de partición en categorías a requisitos expresados en forma de casos de uso y sistemas orientados a objetos. A la hora de aplicar esta propuesta, PLUTO no ofrece nada adicional al propio método de partición de categorías.

El método de partición de categorías es un buen método, como demuestra el hecho de que lo utilice también TDEUML, pero se basa mucho en el dominio del problema y la experiencia de los ingenieros a la hora de identificar las categorías y elecciones. Tampoco es sencillo establecer las restricciones entre categorías y elecciones.

La carencia de herramientas de libre acceso limita la aplicación tanto de PLUTO como de CP.

5.2.6. SCENT

Esta propuesta describe con mucho detalle el proceso de elicitación de requisitos. Sin embargo, con muy poco detalle el proceso de generación de pruebas, por lo que ha sido difícil aplicarla sobre el caso práctico de la sección 4. Es necesario realizar un trabajo muy extenso, 16 pasos, con los escenarios antes de poder generar casos de prueba.

La documentación es inconsistente. El proceso de pruebas descrito en [Glinz99] no coincide con el proceso de prueba descrito en [Ryser03], lo que dificulta su aplicación.

5.2.7. TOTEM

No hay herramientas para automatizar ninguna de las tareas del proceso, aunque sus autores defienden que es una propuesta automatizable mediante herramientas software. Esto es un problema, por ejemplo, en la generación de secuencias de casos de uso, donde obtenemos una explosión combinatoria muy elevada con todas las posibles combinaciones. Este problema se describió en el punto 2.4.

TOTEM no ofrece criterios precisos de selección para reducir el número de secuencias. La propuesta propone para paliar esto generar pruebas aleatoriamente.

Su principal problema es que, aunque ha sido posible obtener resultados, no es una propuesta completa ni terminada.

5.2.8. RBT.

La documentación encontrada sobre esta propuesta es tan pobre que ha sido imposible aplicarla al caso práctico de la sección 4. Esta propuesta está basada en una herramienta comercial que trabaja a partir de diagramas de causa-efecto. No ha sido posible encontrar la suficiente documentación como para poder aplicarla sin contar con la herramienta.

Nos hemos puesto en contacto con la empresa propietaria de la herramienta solicitando una versión de evaluación o educativa de la herramienta pero no hemos obtenido ninguna respuesta.

5.2.9. EUC.

En la aplicación práctica de esta propuesta hemos comprobado que es muy similar al método de partición en categorías. Por ejemplo, las variables operaciones en EUC corresponden con las categorías, los dominios corresponden con las elecciones, las relaciones operacionales corresponden con las restricciones, etc. Sin embargo esta propuesta está menos elaborada y menos detallada.

Por este motivo, EUC no aporta nada que no se encuentre ya en CP y en actualizaciones aplicadas a casos de uso y sistemas orientados a objetos como PLUTO o TDEUML.

5.2.10. AT.

La documentación de esta propuesta es farragosa, ambigua y poco consistente, como se ha expuesto en su descripción en el punto 3.10. Ha sido muy difícil aplicarla al ejemplo práctico y no se han podido obtener resultados.

La pobreza de la descripción nos impide entender, por ejemplo, la diferencia entre los árboles de escenarios y las máquinas de estados, así como las ventajas de desarrollar primero árboles de escenarios y después construir máquinas de estados.

La notación de máquina de estados que utiliza no permite definir de una manera tan clara como los diagramas de estados definidos en UML el evento que lanza la transición y el resultado de dicho evento.

Esta es la propuesta más antigua de todas las analizadas, si exceptuamos a CP.

5.2.11. UCDTC.

No aporta nada que no aporten otras propuestas. Su funcionamiento es similar a TCUC, pero la documentación es más pobre.

5.2.12. TDEUML.

Esta propuesta también está basada en la partición de categorías. Sin embargo, al estar basada en una herramienta comercial hemos encontrado los mismos problemas que con RBT. Sin embargo, la documentación existente es mucho mejor que la documentación sobre RBT.

También nos hemos puesto en contacto con la empresa propietaria de la herramienta solicitando una versión de evaluación o educativa. En este caso nos respondieron que lo iban a consultar pero no hemos obtenido una respuesta firme a día de hoy.

5.2.13. Resumen de los casos prácticos.

Una valoración global, a modo de resumen, de la aplicación de cada propuesta al ejemplo de la sección 4 se muestra en la tabla 103. Esta valoración se realiza a partir de nuestras experiencias en la aplicación de las propuestas.

Tabla 103. Valoraciones de las propuestas en el caso práctico.

Valoración	Propuestas
Propuesta más rápida de aplicar	TCUC, UCPA

Propuesta más lenta de aplicar	TOTEM, PLUTO
Propuesta más sencilla de aplicar	TCUC
Propuesta más compleja de aplicar	AT
Casos de prueba más detallados	TOTEM
Casos de prueba menos detallados	ATCGDM, UCDTC
Propuesta menos detallada	AT, SCENT, UCDTC, TCUC
Propuesta más detallada	TOTEM
Propuestas que no se han podido aplicar en su totalidad	AT, RBT, TDEUML, PLUTO., RETNA

En esta sección se ha mostrado la situación actual, evaluando el nivel de las propuestas a partir de su descripción y a partir de su aplicación en un caso práctico. En la siguiente sección se analizan las conclusiones del análisis realizado en esta sección.

6. Resultado del análisis.

En esta sección se exponen las conclusiones extraídas del análisis realizado en las secciones 3, 4 y 5. Las conclusiones obtenidas se resumen en la tabla 104 y se describen con más detalle en los siguientes párrafos.

Tabla 104. Conclusiones del análisis comparativo.

- No existe una propuesta completa.
- Escaso conocimiento del problema.
- La documentación, en general, es bastante pobre.
- Propuestas aisladas.
- Escasas referencias de aplicaciones prácticas.
- Ninguna medida de su eficacia.
- Falta de estandarización y automatización
- Necesidad de dos estrategias para la generación de casos de prueba.

La primera conclusión obtenida a partir del análisis del estado del arte realizado es que ninguna propuesta está completa. Esto significa que ninguna propuesta ofrece todo lo necesario para generar un conjunto de pruebas del sistema ejecutables sobre el sistema bajo prueba a partir de la especificación funcional del mismo.

Como se ha visto en la descripción de cada propuesta y en el análisis comparativo de la sección 5, la mayoría de las propuestas tienen algunos puntos fuertes. Sin embargo, todas las propuestas presentan algún tipo de carencia que dificulta su puesta en práctica. Estas carencias se muestran en las filas que contienen un “no” en la tabla 102.

También hemos detectado que, en general, las propuestas realizan un estudio incompleto del problema que quieren resolver. Muchas propuestas no contemplan elementos que consideramos fundamentales en cualquier propuesta de generación de prueba, como la generación de valores concretos de prueba o la evaluación del grado de cobertura de los requisitos. Los elementos que consideramos fundamentales se

describen con más detalle en el punto siguiente, como parte del futuro proyecto de investigación.

A título de ejemplo podemos citar la falta de información para la generación de valores de prueba en TCUC o UCPA o la falta de criterios para generar pruebas basadas en casos de uso aislados en la propuesta RBC. La única excepción a este hecho es la propuesta TOTEM, la cual es la más completa de todas las analizadas. Sin embargo, como sus propios autores reconocen, y se ha indicado en su descripción, esta propuesta no está completa. TOTEM no describe todos los pasos de los que consta su proceso de generación de pruebas.

La carencia de documentación también es un hecho importante. Como se ha visto en la tabla 102, la mayoría de las propuestas tienen una calidad media en su documentación. Esto significa qué, a la hora de ponerlas en práctica, aparecen situaciones o dudas no resueltas por la documentación. Esto ha quedado de relieve en la sección 4. Estas situaciones o dudas deben ser resueltas a partir de la experiencia, conocimientos o intuición del equipo humano que las aplique, lo cual les resta efectividad.

En general, ninguna propuesta parte de otras propuestas. Las propuestas incluidas en este trabajo han sido desarrolladas de manera aislada, sin estudiar el estado del arte, investigar las propuestas ya existentes y sin aprovechar los puntos fuertes y corregir los puntos débiles de otras propuestas. Esto queda patente en un conjunto de propuestas similares, como TCUC, EUCP o UCPA que están basadas en el análisis de caminos y prácticamente no aportan nada nuevo respecto a las demás. Una excepción a este hecho supone PLUTO y TDEUML, las cuáles parten de los trabajos existentes sobre particiones en categorías [Ostrand88].

Una carencia grave detectada es la falta de consistencia de los resultados en las propuestas analizadas. La mayoría de las propuestas, en sus resúmenes o títulos prometen que permiten obtener pruebas. Tal y como se han definido las pruebas en la sección 1 de este trabajo, esto consiste en obtener un conjunto de valores de prueba, un conjunto de interacciones con el sistema y un conjunto de resultados esperados. Sin embargo, al final de la mayoría de las propuestas, como se puede ver en la sección 4 y en la tabla 102, obtenemos un conjunto de tablas según el formato particular de cada propuesta. En ninguna propuesta se obtienen pruebas directamente ejecutables (salvo TDEUML, PLUTO y RBT pero contando con las herramientas adecuadas), ni contemplan la generación de los resultados esperados.

Ninguna propuesta incluye referencias ni información sobre cómo evaluar la calidad de las pruebas generadas, excepto EUC. Todas las propuestas parten de la premisa de que, si se aplican adecuadamente sus pasos, se obtiene un conjunto de pruebas con la máxima calidad o cobertura de los requisitos.

Un aspecto que consideramos muy importante y que está poco o nada recogido en las propuestas estudiadas, es la automatización del proceso de generación de pruebas mediante herramientas software. El análisis pone de relieve que, en general, no es posible una automatización completa debido a que los requisitos están expresados en lenguaje natural. Cada una de las propuestas tiene un grado de automatización distinto, en función del detalle en que se definen sus pasos y el uso de formalismos y modelos.

Muchas de las propuestas no siguen ningún estándar, por ejemplo el estándar IEEE para el formato de requisitos o casos de uso, o modelos ampliamente aceptados como los modelos propuestos por UML.

Un problema donde no existe consenso entre las propuestas es cómo resolver los bucles que suelen aparecer en los modelos. Un ejemplo de bucle es, por ejemplo, cuando un usuario introduce un nombre o clave erróneo para autentificarse y el sistema lo vuelve a solicitar. Otro ejemplo es cuando un usuario introduce un código de producto erróneo y el sistema lo solicita de nuevo. En el caso de bucles infinitos, las propuestas suelen dejar el número de vueltas a considerar al criterio de los ingenieros, lo cual dificulta la automatización.

Se ha detectado qué, para que el conjunto de pruebas verifique adecuadamente un sistema es necesario que dichas pruebas se generen siguiendo dos estrategias diferentes. Por una parte es necesario generar pruebas que permitan evaluar todas las posibles secuencias de interacciones descritas en un caso de uso. Por otra parte, es necesario completar dicho conjunto de pruebas con pruebas que verifiquen la ejecución de secuencias de casos de uso. Como se ha expuesto en la descripción de las propuestas y en su análisis, muy pocas de las propuestas analizadas contemplan estas dos estrategias.

7. Proyecto de investigación.

En esta sección expondremos cómo utilizar toda la información obtenida a partir de este trabajo de investigación, para el planteamiento de un nuevo proyecto de investigación.

A partir del análisis de las propuestas, se han identificado una serie de puntos comunes a todas las propuestas. Esto nos permite definir un perfil con todos los elementos que una propuesta de generación de pruebas de sistema debe, al menos, desarrollar. Los puntos comunes se muestran en la tabla 105.

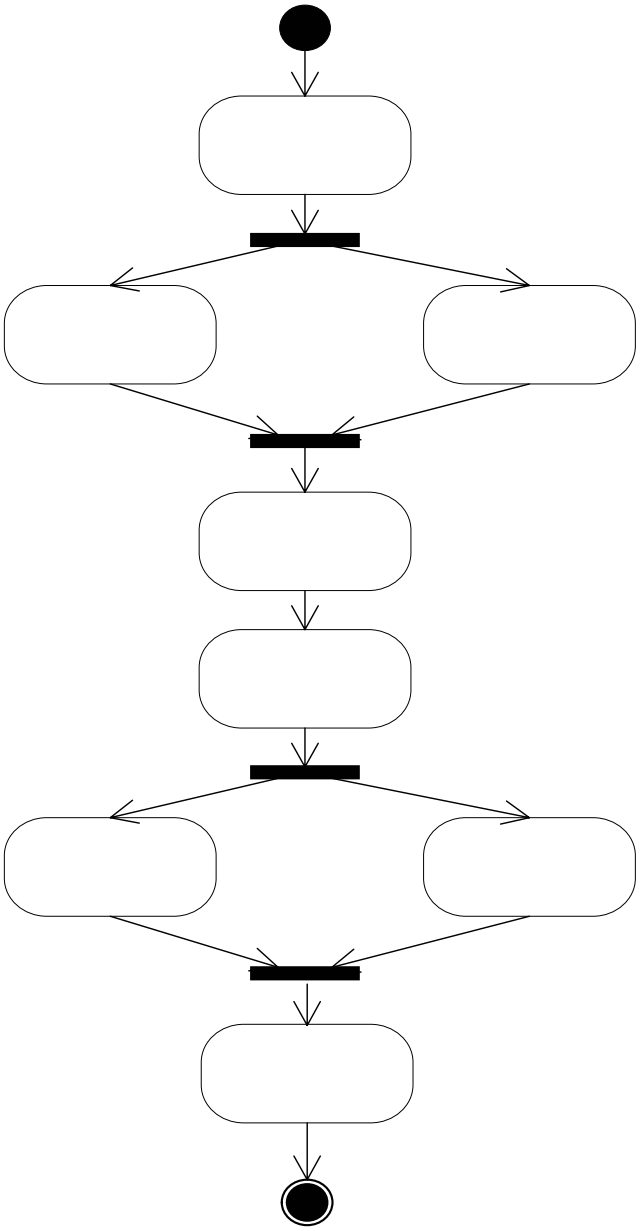
Tabla 105. Puntos comunes de las propuestas de generación de pruebas del sistema.

1	El objetivo de estas propuestas es obtener un conjunto completo de pruebas del sistema que permitan garantizar que el sistema software cumple con la especificación funcional dada, lo cuál permite asegurar su calidad.
2	Todas parten de los requisitos funcionales del sistema y la mayoría permiten comenzar a desarrollar los casos de prueba del sistema en cuanto se comience a disponer de los requisitos funcionales.
3	Todas usan el análisis de los caminos posibles o estados posibles (Salvo PLUTO, TDEUML y EUC).
4	Los requisitos funcionales no tienen que cumplir ningún requisito formal. A partir de una breve descripción en lenguaje natural ya se puede comenzar a trabajar.
5	La generación de pruebas del sistema a partir de los requisitos funcionales se puede realizar de manera automática y sistemática. Casi todas las propuestas son susceptibles de automatización, en mayor o menos medida, 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.

A partir de los resultados del análisis, de los puntos comunes de la tabla 105 y de las carencias puestas de manifiesto en la sección 5 y 6, se han definido las actividades que debe realizar toda propuesta de generación de pruebas del sistema a partir de sus especificaciones funcionales.

Estas actividades se muestran en el diagrama de la ilustración 35 y se describen en los siguientes párrafos.

Ilustración 35. Actividades para la generación de pruebas del sistema.



Como se ha visto a lo largo de este trabajo, las pruebas de sistema tienen como objetivo verificar que el sistema ha implementado los requisitos recogidos en su especificación funcional. Por tanto los requisitos deben ser el punto de partida en la construcción de pruebas del sistema.

La generación del modelo del sistema a partir de su especificación funcional consiste en la construcción de un modelo formal, o semiformal que exprese el comportamiento esperado del sistema en función de la información recogida en sus requisitos. Puesto que el punto de partida son requisitos expresados en lenguaje natural, la construcción de este modelo no puede realizarse de manera automática. Sí será

posible realizar esta actividad de manera sistemática siguiendo un conjunto de pasos claramente definidos.

En la actividad de aplicación del criterio de cobertura, se selecciona un criterio para generar casos de prueba y se aplica sobre el modelo generado en la actividad anterior. El criterio más utilizado en las pruebas estudiadas es el análisis de todos los posibles caminos de ejecución, sin embargo es posible definir otros criterios diferentes, como expone la propuesta RBC. Este proceso puede realizarse de manera automática mediante una herramienta software. También son necesarios criterios que permitan obtener secuencias de casos de uso, como se ha expuesto en las conclusiones del análisis y en la tabla 104.

Como se comentó en la sección 1, uno de los elementos que forman parte de una prueba son los valores o entradas del sistema. Cualquier propuesta completa de generación de casos de prueba debe contemplar la generación de estos valores. La actividad de selección de valores de prueba concretos y representativos es otro de los puntos que se describe de manera muy pobre en las propuestas analizadas.

A partir de los caminos de ejecución y de los valores de prueba es posible construir escenarios de prueba. Un escenario de prueba es una descripción abstracta de un caso de prueba.

El objetivo de la actividad refinado de escenarios de prueba es reducir el número de escenarios de prueba sin afectar a la calidad de los mismos. Esta tarea puede abordarse en la generación de caminos de ejecución buscando criterios de cobertura que minimicen el número de caminos encontrados. Sin embargo, como se ha visto en la sección 1, los valores de prueba pueden aumentar el número de casos de prueba. Un mismo caso de prueba puede ser ejecutado varias veces con valores de prueba distintos. Por este motivo es adecuado tratar la reducción de casos de prueba después de la generación de caminos y de la selección de valores de prueba.

Como se ha visto en el apartado 1, otro de los elementos imprescindibles en una prueba es el resultado esperado. La actividad de generación de resultados tiene como misión determinar la respuesta esperada del sistema ante la ejecución de una prueba. Los resultados esperados pueden obtenerse mediante la aplicación de las pruebas generadas a simulaciones del modelo de comportamiento del sistema de una forma totalmente automática.

La actividad de validación del conjunto de pruebas contempla la posibilidad de evaluar la calidad de las pruebas generadas, por ejemplo midiendo el grado de cobertura

de los requisitos, como propone EUC. Si el nivel de calidad no es el adecuado, puede repetirse de nuevo el proceso cambiando la selección de criterios o cambiando el refinamiento de escenarios de prueba.

Por último, para conseguir un proceso completo es necesario que el resulta final sean pruebas ejecutables sobre el sistema a probar. El paso de generación de código de prueba, aunque se comenta en algunas de las propuestas, no es abordado en detalle por ninguna. El principal impedimento existente en las propuestas analizadas es que ninguna genera un conjunto de pruebas expresadas en un formalismo fácilmente traducible a código. Todas las propuestas analizadas generan casos de prueba expresados en lenguaje natural (salvo las que disponen de herramientas que permiten generar pruebas ejecutables).

A continuación, en la tabla 106, se muestra cada una de las actividades de la ilustración 35, y las propuestas que ofrecen información sobre cómo llevarla a cabo. Algunas actividades solo se mencionan de pasada, o como futuros trabajos en algunas de las propuestas.

Tabla 106. Cobertura de las actividades por parte de las propuestas analizadas.

Actividad	**Propuesta**
1. Generación del modelo del sistema	ATCGDM, UCPA, SCENT, TOTEM, RBT
2. Aplicación del criterio de cobertura	ATCGDM , RBC, UCPA, TCUC, SCENT, RBT
3. Selección de valores de prueba	PLUTO, se nombra solamente en UCPA, EUC.
4. Obtención de escenarios de prueba.	Todas
5. Refinado de escenarios.	UCPA, TOTEM.
6. Generación de resultados	Se nombra solamente en TOTEM.
7. Validación del conjunto de pruebas.	Se nombra solamente en EUC.
8. Generación de código de prueba.	Se nombra solamente en RBC.

A raíz de estas conclusiones, hemos detectado que existen los suficientes motivos que justifiquen la descripción de una nueva propuesta de generación de casos de prueba del sistema a partir de la especificación funcional del mismo. A continuación, se resumen los objetivos del proyecto de investigación a desarrollar a partir de las conclusiones de la sección anterior y de las actividades descritas en la ilustración 35.

En nuestro futuro proyecto de investigación nos planteamos como objetivos principales resolver dos cuestiones.

El primer objetivo del proyecto de investigación será desarrollar una nueva propuesta tomando como base los puntos comunes identificados en la tabla 105. Esta propuesta se beneficiará de todo el trabajo ya hecho, tanto de las propuestas existentes, como del análisis y conclusiones presentados en este trabajo. Esta propuesta incluirá y desarrollará todas las actividades expuestas en la ilustración 35, corrigendo todas las carencias detectadas en las propuestas analizadas y potenciando sus puntos fuertes. En el desarrollo de cada una de las actividades se detallará un conjunto de reglas claras y precisas para que puedan ser aplicadas de una manera sistemática.

Hemos identificado una serie de problemas o cuestiones a resolver para cada una de las actividades de la ilustración 35. Consideramos que estas cuestiones no están adecuadamente resueltas en ninguna de las propuestas. Nuestro futuro trabajo de investigación deberá abordarlas y revolverlas. A continuación, en los siguientes párrafos, se describen algunas de estas cuestiones.

Para la tarea de la generación del modelo de comportamiento del sistema, la primera cuestión a resolver será qué información debe contener la especificación funcional de un sistema para poder generar un modelo de comportamiento y poder generar pruebas del sistema a partir de él. Esta cuestión podemos responderla estudiando la información que piden las propuestas analizadas y viendo que utilidad cumple dicha información dentro del proceso de generación de pruebas.

También hemos de decidir cuál va a ser el modelo de plantilla a utilizar para describir los requisitos funcionales en lenguaje natural. En función de información necesaria, se evaluará si se puede tomar un modelo ya existente, por ejemplo [Cockburn00], o será necesario desarrollar un nuevo modelo de plantillas.

Un aspecto que ninguna propuesta aborda es el grado de detalle de los casos de uso. En general, los casos de uso no son fijos e inmutables. Durante el desarrollo del sistema los casos de uso suelen ser refinados [Dustin02]. Además, durante el desarrollo del sistema aparecen nuevos casos de uso más detallados. Será necesario determinar cuál es el grado de detalle adecuado de los casos de uso para generar pruebas del sistema.

Todas las propuestas analizadas se basan únicamente en los requisitos funcionales del sistema. Alguna menciona de pasada otros tipos de requisitos. Nosotros pensamos que es importante incorporar al proceso otros tipos de requisitos como, por ejemplo, los requisitos de almacenamiento. Teniendo en cuenta los requisitos de almacenamiento, esperamos que sea posible desarrollar valores de prueba de una manera más sencilla y precisa.

También hemos de evaluar si es más adecuado construir un único modelo del sistema, o generar varios submodelos. En las propuestas estudiadas, se construye un único modelo del sistema para generar secuencias de casos de uso, y se construye un modelo por cada caso de uso para generar interacciones posibles a partir de dicho caso de uso.

Según las propuestas analizadas, creemos que el modelo de estados de UML es el diagrama más adecuado para expresar el comportamiento del sistema descrito en su especificación funcional.

Otro aspecto a estudiar es si el modelo del sistema debe ser construido en exclusiva para la generación de pruebas o pueden utilizarse los modelos elaborados en la fase de análisis. Si se pudieran utilizar los modelos de la fase de análisis, tendríamos que plantearnos si merece la pena retrasar el comienzo del proceso hasta obtener esos modelos.

Para la actividad de la aplicación del criterio cobertura, un aspecto a estudiar es cómo utilizar la prioridad de los casos de uso que incorporan algunas propuestas de ingeniería de requisitos, como [Escalona04], para que sea tenido en cuenta por el criterio a la hora de generar interacciones o secuencias.

En general, el criterio de cobertura más ampliamente utilizado es el criterio del cartero chino (Chinese postman) [Robinson00]. Este criterio consiste en recorrer todas las posibles transiciones del sistema con el menor número de caminos posibles.

Sin embargo queremos estudiar en nuestro proyecto de investigación otros posibles criterios de cobertura que permitan, por ejemplo, favorecer el camino principal de ejecución en detrimento de los opcionales. O, por ejemplo, favorecer los caminos que se van a ejecutar más frecuentemente, como propone [Riebisch03].

También queremos evaluar estos criterios con respecto a la generación aleatoria de recorridos, para justificar qué es más eficiente aplicar un criterio.

Se ha visto cómo es necesario generar dos conjuntos de pruebas: un conjunto a partir de las interacciones de cada caso de uso y otro conjunto a partir de secuencias de casos de uso. Será necesario evaluar si en ambos casos pueden utilizarse los mismos criterios de cobertura o cada conjunto necesita sus propios criterios de cobertura.

Para la actividad de selección de valores de prueba pensamos aplicar CP. Nuestro objetivo es conseguir un conjunto de reglas más concretas y sistemáticas a la hora de aplicar CP, reduciendo el número de decisiones que los ingenieros de pruebas tengan que tomar. Como hemos comentado en párrafos anteriores, pretendemos conseguir esto

mediante la incorporación de los requisitos de almacenamiento al proceso de generación de casos de prueba.

Como se ha visto en el análisis, las propuestas estudiadas en este trabajo están centradas en la búsqueda de caminos o en la identificación de categorías. Dado que nuestra propuesta intentará unir ambas aproximaciones, deberá tener en cuenta las pruebas generadas por análisis de caminos y las pruebas generadas por identificación de categorías a la hora de generar los escenarios de prueba.

Una duda que ninguna propuesta nos ha resuelto es la siguiente: si tenemos un camino de ejecución que hemos de recorrer con n valores de prueba distintos, ¿cuántas pruebas tenemos?. Nosotros hemos decidido que un escenario de prueba es un camino de ejecución con un conjunto concreto de valores. Nuestra respuesta a la pregunta anterior es: n pruebas.

En la actividad 4, nuestro proyecto de investigación describirá como unir los caminos de ejecución con los valores de prueba. Se generará un escenario de prueba por cada posible combinación.

Es imprescindible que los escenarios de prueba estén expresados de una manera formal e independiente de las características del sistema y del entorno de prueba. Con esto se facilita la tarea de generar casos de prueba ejecutables para un lenguaje y un sistema en concreto de forma automática. Nuestro proyecto de investigación deberá estudiar todos los formalismos existentes y decidir cuál es el más adecuado para expresar escenarios de prueba. Las características a evaluar van a ser, principalmente, tres: sencillez de uso, flexibilidad y facilidad para generar código a partir de él. Algunos candidatos son el lenguaje TSL o una gramática de etiquetas XML.

A pesar de aplicar un buen criterio de cobertura, es probable que aparezcan pruebas repetidas. Para optimizar el proceso de prueba es necesario eliminar esas pruebas repetidas. Actualmente, de manera preliminar, hemos definido que una prueba A es una prueba repetida respecto de una prueba B cuando se cumple que:

1. A y B tienen los mismos valores de prueba. Esto incluye también las mismas precondiciones y estados iniciales del sistema.
2. A y B tienen el mismo resultado esperado.
3. Las interacciones de A, o la secuencia de casos de uso de A, son iguales o están contenidas en las interacciones, o secuencias de casos de uso, de B.

Con este criterio, A sería una prueba redundante que podría ser eliminada sin afectar a la calidad del conjunto de prueba.

Nuestro proyecto de investigación deberá estudiar si este criterio es siempre válido y si existen otros criterios mejores.

La actividad de generar los resultados de las pruebas consiste en construir oracles. Construir oracles y generar los resultados esperados de los casos de prueba es uno de los mayores problemas en la generación automática de pruebas [Binder00].

El oracle perfecto es aquel que es capaz de proporcionar el resultado válido para cualquier conjunto de entradas. Si fuera sencillo construir un oracle perfecto, también sería sencillo construir un sistema perfecto y las pruebas de sistema no serían necesarias.

En el proyecto AGEDIS, mencionado en el punto 3.13, han resuelto el problema del oracle creando una herramienta que es capaz de construir un prototipo del sistema a partir de su especificación. Esta especificación está compuesta, principalmente, por diagramas de clases y diagramas de estados. Todas las pruebas se aplican a este modelo y, sus resultados, se consideran los resultados esperados del sistema.

El problema principal que presenta esta aproximación es que es necesario tener una especificación detallada del sistema. Esto obliga a retrasar el proceso de generación de pruebas hasta las fases de análisis o de diseño.

El futuro proyecto de investigación deberá buscar otras alternativas y evaluar la más adecuada. Si finalmente la solución adoptada es la utilizada en el proyecto AGEDIS, será necesario evaluar la posibilidad de utilizar sus herramientas.

Como se ha comentado en este trabajo, la mayoría de las propuestas no permiten comprobar la calidad del conjunto de pruebas generado. Es razonable suponer que un ingeniero de pruebas puede no haber escogido el criterio más adecuado, o haber olvidado categorías de datos importantes. También es razonable plantearse que las herramientas de soporte pueden presentar fallos, como en el caso de RBC.

Por estos motivos, consideramos imprescindible que el futuro proyecto de investigación permita validar el conjunto de pruebas generado. Esta validación puede estar basada, por ejemplo, en la cobertura de los requisitos por parte de las pruebas. Si la cobertura de los requisitos es la deseada, el conjunto de pruebas se considera válido. Si no, será necesario repetir el proceso de generación.

Otra cuestión que el proyecto de investigación deberá resolver es como cuantificar el grado de cobertura de los requisitos por parte de las pruebas.

Consideramos muy importante expresar los escenarios de prueba de manera formal para favorecer la generación de código de prueba. El proyecto de investigación debe

estudiar como ejecutar este código de prueba contra la interfaz externa del sistema. Por ejemplo, puede ser posible generar código para un robot que trabaje con un sistema como si fuera un usuario humano.

El segundo objetivo del proyecto de investigación será el estudio de la viabilidad de la automatización de cada una de las actividades de la nueva propuesta y la implementación de dichas actividades en una herramienta software que permita la automatización real del proceso de generación de casos de prueba. Por último se desarrollará un prototipo de esta herramienta.

8. Foros relacionados.

En esta sección se describen todos los foros encontrados que, en la actualidad, desarrollan trabajos de investigación relacionados con la generación de casos de prueba de sistema a partir de la especificación funcional del sistema en desarrollo.

8.1. Publicaciones relevantes.

Una de las publicaciones más importantes en el ámbito de las pruebas del software es The Journal of Software Testing, Verification and Reliability [STJ]. Esta publicación está orientada al ámbito académico y puede ser consultada libremente a través de Internet.

Otra publicación relevante es Better Software Magazine [BetterSoftware], la cuál trata aspectos de pruebas y calidad del software. Esta publicación, a diferencia de la anterior, está más centrada en el ámbito empresarial e industrial que en el ámbito académico. Muchos de sus artículos pueden consultarse de forma gratuita en la web de StickyMinds [StickyMinds].

También hay que reseñar la publicación web The Journal of Software Testing Professionals [JournalSoftware], una publicación en formato electrónico gestionada por The International Institute for Software Testing y que también está más orientada al mundo empresarial que al mundo académico.

Como se puede apreciar consultando las referencias de las propuestas, el ámbito de publicación de trabajos de este tipo no se limita sólo a publicaciones restringidas al ámbito de la prueba del software, sino que son aceptadas en publicaciones relacionadas con la ingeniería del software, la calidad de los sistemas, etc. Algunos ejemplos son PLUTO, publicado en Lecture Notes in Computer Science, o AT, publicada en Anuals of Software Engineering.

8.2. Congresos.

Al estudiar los congresos centrados en generación de pruebas encontramos la misma situación comentada en el punto sobre publicaciones. Existen pocos congresos

específicos sobre prueba del software. Sobre generación de casos de prueba en concreto, no hemos encontrado ninguno. Sin embargo los trabajos sobre prueba del software suelen ser aceptados en un amplio conjunto de congresos relacionados con la ingeniería del software y la calidad de los sistemas software.

A continuación se detallan una serie de congresos relevantes para investigadores de las pruebas del software.

International Symposium on Software Testing and Analysis (ISSTA)

ISSTA es un simposio que se celebra cada dos años y se autodefine como la conferencia líder en el campo de la prueba y análisis del software. El índice de trabajos aceptados en este congreso no suele superar el 25%. El enlace a la próxima convocatoria, prevista para el año 2.006, se muestra a continuación.

http://www.cis.udel.edu/issta06/

International Workshop on test and Analysis of Component-based Systems (TACOS)

http://www.lta.disco.unimib.it/tacos/workshop.htm

Workshop on system Testing and Validation

Este taller está centrado en mecanismos de prueba y validación de sistemas, dentro de los cuales tiene cabida las pruebas del sistema.

http://syst.eui.upm.es/conference/sv04/

StarWest, StarEast y Better Software Conference.

Estas conferencias se organizan cada año en distintas partes de Estados Unidos. Están más centradas en el mundo empresarial que en el mundo académico. De la conferencia del año 2002, surgió la propuesta UCPA. A continuación se incluyen los enlaces a las páginas de estas conferencias.

http://www.sqe.com/starwest/

http://www.sqe.com/stareast/

http://www.sqe.com/bettersoftwareconf/

ICSTEST

Este congreso es la versión española de las conferencias del párrafo anterior. Tiene una repercusión internacional y está más orientado al mundo de la industria que al mundo académico.

http://www.sqs.es/icstest/

Otros congresos no relacionados directamente con las pruebas software donde este tipo de propuestas tienen cabida y donde se han expuesto trabajos referenciados en este informe son:

CAiSE y CAiSE Forum

http://www.fe.up.pt/caise2005/

European Software Engineering Conference

http://esecfse05.di.fct.unl.pt/home.htm

European Conference on Object-Oriented Programming

http://www.ifi.uio.no/ecoop2004/

IEEE International Requirements Engineering Conference

www.re04.org

www.re05.org

IEEE International Symposium on Software Reliability Engineering

http://rachel.utdallas.edu/issre/

The IASTED International Conference on SOFTWARE ENGINEERING

http://www.iasted.org/conferences/2005/

Algunos congresos españoles donde tienen presencia trabajos sobre pruebas son:

I Simposio Sobre Avances En Gestión De Proyectos Y Calidad Del Software

Sin dirección web conocida

JISBD

http://cedi2005.ugr.es/isydb

8.3. Centros de investigación.

En este punto se recogen los centros de investigación que, actualmente, desarrollan proyectos relacionados con la generación de pruebas o pruebas del software en general, y centros de investigación que cuenta con investigadores trabajando en este área.

International Institute for Software Technology.

http://www.iist.unu.edu/

Este instituto está localizado en Macao y pertenece a la universidad de las Naciones Unidas. Actualmente, aunque desarrolla trabajos sobre generación de casos de prueba, éstos se centran más en especificaciones formales a partir de su propio lenguaje de especificación formal RAISE [RAISE95].

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"

http://www.iist.unu.edu

Este instituto cuenta con un laboratorio de ingeniería del software. Uno de los focos de investigación de este laboratorio es la prueba de grandes sistemas. De hecho, la directora de este laboratorio, Dña. Antonia Bertolino, ha sido la desarrolladora de PLUTO y varios trabajos más en el campo de las pruebas del software y la generación de pruebas referenciados en este informe.

Fraunhofer Institute Computer Architecture and Software Technology

http://www.first.fraunhofer.de

Este instituto se encuentra en Alemania y ha desarrollado trabajos importantes en el área de generación de pruebas del sistema. Uno de estos trabajos es el único informe que hemos podido encontrar sobre propuestas para generación de casos de prueba a partir de especificaciones funcionales [Quasar03], aparte del presentado en este trabajo.

Según las conclusiones de dicho trabajo, este instituto está trabajando en una nueva propuesta de generación de casos de prueba del sistema. Sin embargo, no hemos podido contactar con ningún miembro de este instituto que nos informe del estado de ese proyecto.

Sí hemos podido contactar con otros investigadores del mismo instituto, en concreto con D. Mario Friske, que están trabajando en la generación de pruebas de sistema "aplicando tecnología del MDA" (según sus propias palabras). Sin embargo, todos sus trabajos están en alemán, por lo que no han podido ser incluidos en este estudio.

8.4. Grupos de investigación.

En este punto destacamos aquellos grupos de investigación que están trabajando en la generación de casos de prueba.

Uno de los grupos más relevantes son los miembros del Institut für Informatik Programmierung und Softwaretechnik de la universidad de Manchen, en Alemania. En concreto, uno de sus investigadores, la doctora Nora Koch, ha sido la principal creadora de UWE [Koch01]. UWE es una propuesta metodológica basada en el Proceso Unificado y UML para el desarrollo de aplicaciones web.

Actualmente este grupo acaba de comenzar un proyecto de investigación para incorporar a UWE la generación automática de casos de prueba a partir de sus modelos.

La dirección del website del grupo se incluye a continuación

http://www.pst.informatik.uni-muenchen.de/

Otro grupo de investigación es el Angewandte Datentechnik de la Universität Paderborn. En concreto uno de los focos de investigación de este grupo es la verificación de software mediante la comprobación de modelos. La dirección del website del grupo se incluye a continuación.

http://adt.upb.de/

Otro grupo de investigación relevante en la generación de casos de pruebas es el Department of Systems and Computer Engineering en Canadá. Miembros de este departamento han elaborado la propuesta TOTEM, descrita en el punto 3.7. La dirección del website del grupo se incluye a continuación

http://www.sce.carleton.ca/

En España no hemos encontrado ningún grupo de investigación que desarrolle trabajos centrados en la generación de pruebas de sistema. Sí hemos contactado con investigadores de la universidad de Extremadura, en concreto con la doctora María del Sol Sánchez-Alonso, que están trabajando en la generación de casos de prueba a partir de requisitos de colaboración [Sanchez01].

En concreto, y según sus propias palabras, sus investigaciones se basan en simular el comportamiento de entornos con importantes restricciones de coordinación, centrándose en el comportamiento coordinado, como forma de probar que los requisitos de coordinación del sistema son válidos. Se trata por tanto de una prueba de alto nivel y que posteriormente han completado con un chequeador de concordancia, haciendo uso del mismo lenguaje formal MAUDE [Clavel99], para garantizar que las especificaciones y la descripción detallada del sistema a lo largo de los sucesivos refinamientos son consistentes.

Si embargo sus trabajos no tienen ninguna aplicación en la generación de pruebas del sistema a partir de la especificación funcional.

Referencias

[Andrews05] A. A. Andrews et-al. 2005. Testing Web Applications by Modeling with FSMs. Software Systems and Modeling 4(2).

[Balcer90] M. Balcer, W. Hasling, and T. Ostrand. 1990. Automatic Generation of Test Scripts from Formal Test Specifications. Proceedings of ACM SIGSOFT'89 - Third Symposium on Software Testing, Verification, and Analysis (TAVS-3), ACM Press, pp. 257-71.

[Bertolino02] Bertolino, A., Fantechi, A., Gnesi, S., Lami, G., Maccari, A. Use Case Description of Requirements for Product Lines, REPL'02, Essen, Germany, Avaya Labs Tech. Rep. ALR-2002-033.

[Bertolino03] Bertolino, A., Gnesi, S. 2003. Use Case-based Testing of Product Lines. ESEC/FSE'03. Helsinki, Finland.

[Bertolino04] Bertolino, A., Gnesi, S. 2004. PLUTO: A Test Methodology for Product Families. Lecture Notes in Computer Science. Springer-Verlag Heidelberg. 3014 / 2004. pp 181-197.

[BetterSoftware] The Better Software Magazine. http://www.stickyminds.com/BetterSoftware/magazine.asp

[Binder00] Robert V. Binder. Testing Object-Oriented Systems. Addison-Wesley. USA.

[Blackburn94] Blackburn P., Bos J. 19994. Working with Discourse Representation Theory: An Advance Course in Computational Semantics.

[Boddu04] Boddu R., Guo L., Mukhopadhyay S. 2004. RETNA: From Requirements to Testing in Natural Way. 12th IEEE International Requirements Engineering RE'04.

[Cavarra04] Cavarra, Alessandra, Davies, Jim 2004 Modelling Language Specification. AGEDIS Consortium Internal Report. http://www.agedis.de/

[Clavel99] Clavel, M. et-al. 1999. Specification and Programming in Rewriting Logic. Computer Science Laboratory. SRI International.

[Cockburn00] Cockburn, Alistair. 2000. Writing Effective Use Cases. Addison-Wesley 1st edition. USA.

[Craggs03] Ian Craggs, Manolis Sardis, Thierry Heuillard. 2003. AGEDIS Case Studies: Model-Based Testing in Industry. AGEDIS Consortium Internal Report . http://www.agedis.de/

[Dan02] Li Dan and Bernhard K. Aichernig. 2002. Automatic Test Case Generation for RAISE. Internal Report. United Nations University International Institute for Software Technology.

[Dustin02] Dustin E. et-al. 2002. Quality Web Systems. Addison-Wesley 1st edition. USA.

[Escalona04] Escalona M.J. 2004. Modelos y técnicas para la especificación y el análisis de la Navegación en Sistemas Software. Ph. European Thesis. Department of Computer Languaje and Systems. University of Seville. Seville, Spain.

[Fikes71] R. E, Fikes, N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 1971.

[Fröhlich00] Fröhlich, P, Link, J. 2000. Automated Test Case Generation from Dynamic Models. ECOOP 2000. pp. 472-491.

[Fröhlich99] Fröhlich, P, Link, J. 1999. Modelling Dynamic Behaviour Based on Use Cases. Proceedings of Quality Week Europe. Brussels.

[Glinz99] Martin Glinz, Johannes Ryser. 1999. A Practical Approach to Validating and Testing Software Systems Using Scenarios Quality. Week Europe QWE'99 in Brussels, Institut für Informatik, Universität Zürich.

[Goed86] Goedel, K. 1986. Über formal unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme. Unvollständigkeitstheoreme. Collected works vol. 1. Oxford University Press. New York. USA.

[Hartman04] Hartman , Alan 2004 AGEDIS Final Project Report AGEDIS Consortium Internal Report. http://www.agedis.de/

[Henriksen95] Henriksen J. G., el-al. 1995. Mona: Monadic Second-Order Login in Practice. TACAS'95.

[Heumann02] Heumann , Jim, 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals.

[Hsia94] Hsia, P, et-al. 1994. A Formal Approach to Scenario Analysis. IEEE Software 11, 2, 33-41.

[Hsia95] Hsia, P, et-al. 1994. A Usage-Model Based Approach to Test the Therac-25. Safety and Reliability in Emerging Control Technologies. IFAC Workshop. Pp55-63. Florida. USA.

[Hsia97] Hsia, P, et-al. 1997. Software Requirements and Acceptance Testing. Anuals of software Engineering. Pag 291-317.

[Itschner98] 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.

[Jalote02] Jalote, Pankaj, 2002. Software Project Management in Practice. Addison Wesley. USA.

[Java04] Varios. 2004. Java Language Specification Third Edition. Sun Microsystems.

[JournalSoftware] The Journal of Software Testing Professionals. http://www.testinginstitute.com/journal.php

[Koch01] Koch, N. 2001. Software Engineering for Adaptative Hypermedia Applications. Ph. Thesis, FAST Reihe Softwaretechnik Vol(12), Uni-Druck Publishing Company, Munich. Germany.

[Labiche02] L. Briand, Y. Labiche. 2002. A UML-Based Approach to System Testing. Carleton University Inner Report.

[Littlewood87] B. Littlewood, ed., Software Reliability: Achievement and Assessment (Henley-on-Thames, England: Alfred Waller, Ltd., November 1987).

[Métrica3] Métrica 3. Ministerio de Administraciones Públicas. http://www.map.es/csi

[Mogyorodi01] Gary Mogyorodi. 2001. Requirements-Based Testing: An Overview. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39). pp. 0286

[Mogyorodi02] Gary E. Mogyorodi. 2002. Requirements-Based Testing: Ambiguity Reviews. Journal of Software Testing Professionals. p 21-24.

[Mogyorodi03] Gary E. Mogyorodi .What Is Requirements-Based Testing?. 15th Annual Software Technology Conference. Apr. 28-May 1. Salt Lake City, USA

[Myers79] Glenford J. Myers. 1979. The art of software Testing. John Wiley & Sons. New York. USA.

[Naresh02] Ahlowalia, Naresh. 2002. Testing From Use Cases Using Path Analysis Technique. International Conference On Software Testing Analysis & Review.

[Nebut03] Nebut, C. Fleurey, et-al. 2003. Requirements by contract allow automated system testing. Procedings of the 14th International symposium of Software Reliability Engineering (ISSRE'03). Denver, Colorado. EEUU.

[Nebut04] Nebut, C. Fleurey, et-al. 2004. A Requirement-Based Approach to Test Product Families. LNCS. pp 198-210.

[Offutt03] Jeff Offutt, Aynur Abdurazik. 2003. Generating Tests from UML Specifications. The Journal of Software Testing, Verification and Reliability 13(1)25-53.

[OCL03] 2003. OMG OCL Specification 2.0. http://www.omg.org

[Ostrand88] Ostrand, TJ, Balcer, MJ. 1988. Category-Partition Method. Comunications of the ACM. 676-686.

[Pankaj02] Jalote, Pankaj, 2002. Software Project Management in Practice. Addison Wesley.

[Pressman04] Pressman, Roger, 2004. Ingeniería del Software. Un Enfoque Práctico. Sexta edición Prentice-Hall.

[Quasar03] Denger, C. Medina M. 2003. Test Case Derived from Requirement Specifications. Fraunhofer IESE Report.

[RAISE92] The RAISE Language Group. The RAISE Specification Language. BCS Practitioner Series.Prentice Hall, 1992.

[RAISE95] The RAISE Method Group. The RAISE Development Method. BCS Practitioner Series. Prentice Hall, 1995.

[RBCTool] UCTSystem. http://www.irisa.fr/triskell/results/ISSRE03/UCTSystem/UCTSystem.html

[Riebisch03] Matthias Riebisch, Ilka Philippow, Marco Götze Ilmenau. 2003. UML-Based Statistical Test Case Generation. Technical University, Ilmenau, Germany

[Robinson00] Robinson, Harry. 2000. Intelligent Test Automation. Software Testing & Quality Engineering. Pp 24-32.

[Ruder04] Axel Ruder et-al. 2004. A Model-based Approach to Improve System Testing of Interactive Applications. ISSTA'04. Boston, USA.

[Ruder04-2] Axel Ruder. 2004. UML-based Test Generation and Execution. Rückblick Meeting. Berlin.

[Ryser03] 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.

[Sanchez01] Sanchez-Alonso, M. Herrero, J.L. 2001. Entono Basado en el Uso de Técnicas Formales para el Desarrollo y Prueba de Sistemas. JISBD. Pp. 189-203.

[STJ] The Journal of Software Testing, Verification and Reliability. http://www3.interscience.wiley.com/cgi-bin/jhome/13635

[StickyMinds] StickyMinds Website. http://www.stickyminds.com

[Swebok04] Varios autores. 2004. SWEBOK. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society.

[Srivastaba97] Srivastava V. K. 1997. Experience with the Use of Standard IEEE 982.1 on Software Programming. IEEE. Pp 121-127

[Vazquez01] PJ Vázquez, et-al. 2001. VERIFICACIÓN CON XML. Informe Interno. Departamento de Informática y Automática. Universidad de Salamanca

[UML03] 2003. OMG Unified Modeling Language Specification 2.0. http://www.omg.org

[Wood02] Wood D., Reis J. 2002. Use Cased Derived Test Cases. StickyMind. www.stickymind.com

[Wu02] Ye Wu and Jeff Outt. 2002 Modeling and Testing Web-based Applications. GMU ISE Technical ISE-TR-02-08.