# Una representación gráfica del testing ágil

# A graphical representation of agile testing

Carlos M. Zapata[1], Ph.D., Adrian S. Arboleda C.[2], Ing., Carlos E. Castrillón V.[3], Adm.
1. Grupo de Lenguajes Computacionales, Escuela de Sistemas, Facultad de Minas, Universidad Nacional de Colombia, Sede Medellín.
2. Grupo T&T, Facultad de Minas, Universidad Nacional de Colombia, Sede Medellín.
3. Estudiante de Especialización en Sistemas, Universidad Nacional de Colombia, Sede Medellín
cmzapata@bt.unal.edu.co - asarbolec@unal.edu.co - ccastrillonv@unal.edu.co

Recibido para revisión 14 de enero de 2010, aceptado 4 de junio de 2010, versión final 8 de julio de 2010

*Resumen*—**Las pruebas ágiles surgieron abarcando los principios del manifiesto ágil, orientado a entregar software de calidad para la satisfacción de las necesidades del interesado. Esta concepción implica utilizar técnicas y prácticas de *testing* adaptadas a métodos de desarrollo ágil. La incorporación del *testing* ágil en los métodos de desarrollo encuentra problemas de comunicación entre los miembros de los equipos de desarrollo y una carencia de automatización de los procesos de pruebas de software, por lo que es necesario explorar instrumentos visuales que contribuyan en la solución de estos problemas. Este artículo propone la representación de los fundamentos del *testing* ágil mediante un esquema preconceptual, donde se pretende extraer el conocimiento base de manera gráfica. El esquema preconceptual resultante, por su sintaxis, facilita la comunicación de las ideas del *testing* ágil entre humanos y máquinas.**

*Palabras Clave:* **Pruebas ágiles, representación gráfica, probador ágil, automatización de pruebas, desarrollo ágil, esquema preconceptual.**

*Abstract*—**Agile testing (AT) has emerged for covering the principles of agile manifesto. Agile testing is oriented to deliver software quality in order to satisfy the stakeholder needs. This issue involves the usage of testing techniques and practices, customized to cover agile development methods. Involving agile testing inside software development methods meet team-member miscommunication problems and lacking of automated software test processes. By this reason, exploring visual tools is necessary for contributing to solve these problems. In this paper we propose the representation of agile testing foundations by using a pre-conceptual schema, which aims to graphically extract basic knowledge. Due to its syntax, the resulting pre-conceptual schema can ease the communication of agile testing ideas among humans and machines.**

*Keywords:* **Agile testing, graphical representation, agile tester, agile development, test automation, pre-conceptual schemas.**

## I. INTRODUCCION

El nuevo milenio trajo consigo nuevas formas de desarrollo de software, algunas de las cuales trabajan más con el interesado en el proceso de elicitación de requisitos, diseño y construcción de software. De una manera que proclama ser más rápida que los métodos tradicionales de desarrollo de software, el desarrollo ágil se presenta como una alternativa para generar software de calidad [1]. Sin embargo, la incorporación del *testing* en el ciclo de vida de los métodos ágiles de desarrollo aún encuentra varias limitaciones, como la mala comunicación en los equipos de desarrollo y la carencia de automatización de las pruebas en diferentes fases del ciclo de vida [2, 3, 4].

Como respuesta a estas limitaciones, se vienen adelantando algunos esfuerzos en representación del conocimiento en *testing* ágil [5, 6], pero aún emplean representaciones que, si bien permiten algún nivel de comunicación, se restringen en partes del proceso o lo abordan de forma general y, adicionalmente, se encuentran lejanas de la automatización que se plantea como solución a los problemas de difusión del *testing* ágil.

La representación del conocimiento, que es una alternativa viable para solucionar este tipo de problemas, requiere un soporte formal para transmitir un dominio específico [7]. Además, se requieren representaciones del conocimiento que puedan interactuar con los humanos y con las máquinas, con el fin de lograr la automatización que se plantea. Los esquemas preconceptuales [8] surgen como herramientas para representar el conocimiento en cualquier dominio, ya que se basan en lógica proposicional. Por ende, son computacionalmente tratables y cualquier persona o máquina puede comprender su sintaxis [8]. Por estas razones se convierten en una alternativa viable para la representación del conocimiento en *agile testing*.

De acuerdo con lo anterior, este artículo propone una representación grafica de *testing* ágil mediante un esquema preconceptual. Se busca orientar a los integrantes de equipos ágiles para desarrollar sus destrezas y conocer los fundamentos del manifiesto ágil hacia la construcción valiosa de un producto de software en conjunto con el interesado, integrando allí las ideas del *testing* ágil como paso inicial para su automatización.

La estructura de este artículo es la siguiente: en la parte II se introduce la historia de la representación del conocimiento dentro del área de las pruebas ágiles; la parte III expone las nociones de representaciones gráficas y las ideas generales de pruebas ágiles; en la parte IV se propone el desarrollo de una representación gráfica por medio de un esquema preconceptual; la parte V incorpora los principios de pruebas ágiles en un esquema preconceptual; finalmente, en la parte VI se presentan las conclusiones y el trabajo futuro del artículo.

## II. ANTECEDENTES

El pensamiento del desarrollo ágil procura generar rapidez en el desarrollo, calidad en el producto y satisfacción del interesado, principalmente. Este concepto implica utilizar nuevas técnicas y prácticas de *testing* adaptadas al nuevo método de desarrollo, donde los valores ágiles deben guiar al probador, al equipo y a la organización [5]. Algunas experiencias de *testing* ágil [2, 3, 4] resaltan ciertas técnicas que se emplean para asegurar la calidad de los productos en distintos proyectos, tales como las pruebas por pares, las pruebas de regresión y la utilización de herramientas adecuadas, como un sistema rastreador de *bugs*. Por otra parte, manifiestan el rol importante del cliente en la realización de pruebas de aceptación. En un equipo ágil de desarrollo de software, todos los miembros son probadores, resaltando la necesidad de hacer pruebas de forma paralela a la programación de una determinada funcionalidad. Sin embargo, todas esas

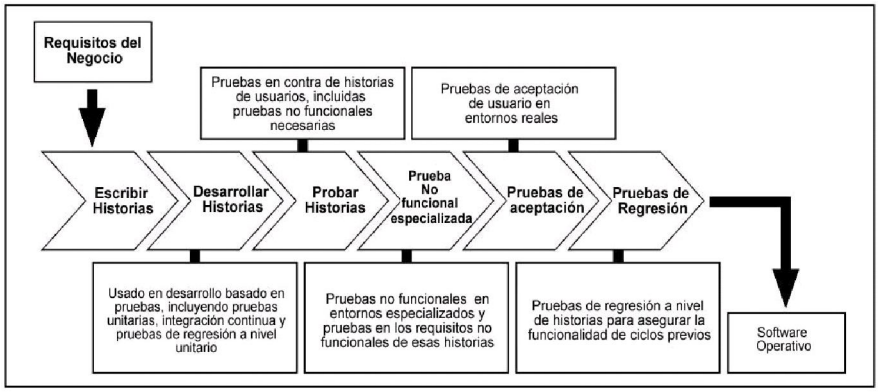


**Figura 1.** Actividades de *testing* al interior de un desarrollo ágil. Tomado de [6]

experiencias tienen algunos problemas en común: la deficiente comunicación con otros integrantes de los grupos de desarrollo, la mala estimación de pruebas y la falta de automatización de las mismas.

Las técnicas de pruebas ágiles son relativamente nuevas, pero ya existe una propuesta [9] que enseña algunos valores para la transformación desde los equipos tradicionales hacia los equipos ágiles por medio de estrategias, planificación de metas y automatización de pruebas clave. Esta propuesta plasma una visualización de los niveles de pruebas de regresión, manifestando utilidad y tiempos de realización. En otra propuesta [6] se muestra, gráficamente, la complejidad del *testing* cuando se combina con metodologías ágiles de desarrollo de software. Esta propuesta, que se puede apreciar en la figura 1, muestra una aproximación gráfica al proceso de *testing* ágil. En esta propuesta se encuentran, nuevamente, los problemas que menciona la implementación del *testing* ágil en una organización: la necesidad de comunicación al interior del grupo de desarrollo y la automatización de pruebas como una forma de facilitar la inserción del proceso en el ciclo de vida del desarrollo ágil.

Crispin y Gregory [5] utilizan los mapas mentales para representar argumentos que rodean al *agile testing*. Las temáticas principales que se exponen allí, como se muestra en la figura 2, son: una introducción a *agile testing* y sus valores, las actividades que rodean este proceso y las diferencias con los equipos tradicionales de desarrollo. Por medio de otros mapas conceptuales, los autores amplían conceptualmente estos y otros tópicos, permitiendo su acercamiento a la representación del conocimiento. Sin embargo, este tipo de mapas, si bien incluyen conceptos y son perfectamente entendibles para transmitir conocimientos entre humanos, no contienen la estandarización de términos necesaria para su tratamiento computacional, pues los contenidos de cada uno de los rectángulos son variables y no permiten establecer una diferenciación clara entre ellos. Esto hace que se requieran otras características de un esquema de representación del conocimiento, tal como se explora en la siguiente sección.

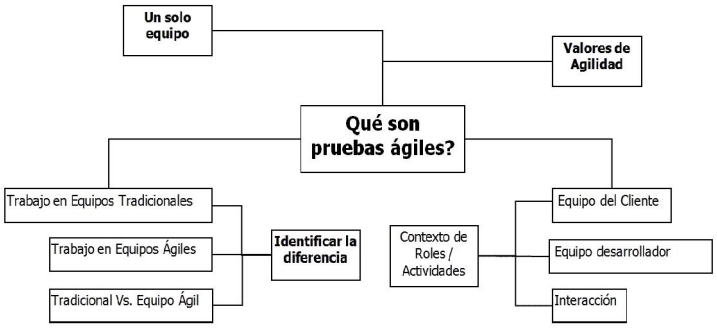


**Figura 2.** Representación agile testing. Tomado de [5]

## III. MARCO TEORICO

### *¿qué es una representación gráfica?*

La representación del conocimiento surgió como un área de la inteligencia artificial entre los años 70 y 80 para representar concepciones sobre un dominio particular, con el fin de alcanzar un razonamiento basado en esa realidad expresada. Lakemeyer y Nebel [7] examinan los fundamentos teóricos para que una representación del conocimiento sea formal y válida para el proceso de razonamiento desde el punto de vista lógico y computacional, identificando, principalmente, tres aspectos: un lenguaje apropiado, la realización de inferencias adecuadas y la evolución del conocimiento desde esa representación. La representación del conocimiento trata el problema del simbolismo, el mantenimiento y la manipulación de conocimiento sobre un dominio de aplicación. Una definición de la representación del conocimiento podría ser una sustitución racional interna, que permite integrar conceptos en un contexto dado. En representación gráfica, el ser humano recuerda en mayor proporción las imágenes o los conceptos cortos en comparación con otras percepciones como sonidos o la misma lectura [10]. Por esta razón, es necesario encontrar nuevos instrumentos visuales que permitan originar nuevo conocimiento. Algunos instrumentos que se suelen usar para la representación del conocimiento son: los mapas conceptuales, los mapas mentales, las redes semánticas y los esquemas preconceptuales, entre otros.

Un mapa conceptual, es un medio de aprendizaje caracterizado por su jerarquización, impacto visual y simplificación de un tema particular, pero que carece de una formalización válida en su representación [10].

Un mapa mental es un método de análisis que permite organizar con facilidad los pensamientos y utilizar al máximo las capacidades mentales. Por otra parte, debido a su sencillez para gestionar el flujo de información entre el cerebro y el exterior, se convierte en un instrumento eficaz para tomar notas y planificar los pensamientos [11]. Sin embargo, todos los mapas mentales tienen algo en común: su estructura natural compuesta por ramas que irradian una imagen central. Los conceptos se enlazan mediante el uso de colores, símbolos, dibujos y palabras, utilizando un conjunto de reglas sencillas y amigables, pero es, precisamente, esta sencillez la que no permite plasmar conceptos que la computación sea capaz de tratar lógicamente,

Los esquemas preconceptuales, por otra parte, se originan a partir de un lenguaje controlado, llamado UN-Lencep (Universidad Nacional—lenguaje controlado para la especificación de esquemas preconceptuales), el cual posee un conjunto básico de plantillas que permite una mejor interacción entre desarrolladores e interesados, en forma de subconjunto del lenguaje natural [8]. Estos esquemas tienen las herramientas básicas de lógica, como equivalencia semántica y condicionales, donde se pueden construir relaciones "es", "tiene" y "si…entonces". El siguiente es un discurso en UN-Lencep (tomado de [8]) sobre una temática de autores de libros, mientras que la figura 3 muestra la representación de ese discurso por medio de un esquema preconceptual.

Libro TIENE ISBN
Libro TIENE Titulo
Libro TIENE Precio
Autor TIENE Nombre
Autor TIENE Dirección
Autor TIENE Fecha_de_nacimiento
Empleado TIENE Nombre
Empleado TIENE Número_de_seguro_social
Editorial TIENE Dirección
Editorial TIENE Empleado
Editorial TIENE Denominación
Editorial PUBLICA Libro
Autor ESCRIBE Libro

La representación del conocimiento debe tener un soporte lógico formal, tal como lo expone Geller [12], quien creó un lenguaje natural mediante formas gráficas, destacando la necesidad de crear un "lenguaje gráfico profundo" basado en la lógica proposicional. De esta manera, se busca controlar la ambigüedad que puede tener un idioma en varios contextos.

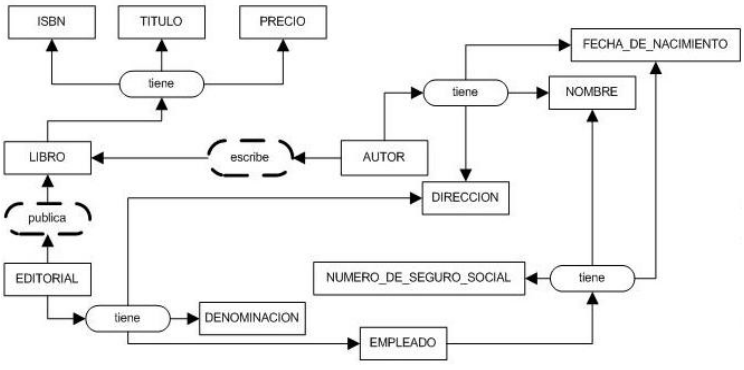


**Figura3.** Esquema preconceptual acerca de la concepción autor-libro. Tomado de [8].

### *¿qué son pruebas ágiles?*

Hacia el año 2001 se presentó el manifiesto ágil [1], que introduce los valores y los principios que deben adoptar los miembros de un equipo ágil para llevar a cabo el desarrollo adecuado y rápido de una aplicación. Entre ellos, se destaca la prioridad de satisfacer al interesado por medio de las entregas tempranas de software, la colaboración continua entre equipo interesado y equipo desarrollador y la mejora continua en el proceso, porque siempre se responde al cambio y a las nuevas expectativas del interesado. Algunas organizaciones enfrentan un cambio cultural al tratar de acoger estas metodologías, por lo cual esta idea no se puede aplicar a todas las organizaciones. Aquellas que tienen un pensamiento independiente, no tienen establecida una estructura jerárquica y tienen políticas prioritarias de calidad, son más adecuadas para asumir un pensamiento ágil [5].

De acuerdo con Crispin y Gregory [5], dentro de un equipo ágil existe comunicación continua entre los desarrolladores e interesados, definiendo los requisitos, las pruebas y el comportamiento esperado del producto. Además, se manifiesta el concepto de roles, donde se intenta que cada miembro posea un buen número de roles para eliminar la dependencia entre individuos, promoviendo siempre a todos el principio de calidad en sus actividades diarias. En este marco, en un equipo ágil todos los individuos tienen habilidades de probadores como establecen Talby *et al.* [4].

Un probador ágil se caracteriza por tener, además de habilidades técnicas en *testing*, destreza para integrar las necesidades del interesado al equipo desarrollador, definiendo las historias del usuario en requisitos y, también, transformándolas en parámetros de prueba cuando se codifiquen. El rol de probador sobresale por colaborar con todos los integrantes del equipo.

Según Crispin y Gregory [5], los conceptos y definiciones que giran alrededor del *testing* ágil son:

***Tester***: Persona del equipo que puede desempeñar varios roles en un proyecto de software. Tiene habilidades y destrezas para el desarrollo y ejecución de pruebas.

***Interesado***: Persona o entidad que se apoya en un analista, con el fin de proponer una solución a un problema. Es quien elabora el discurso de los requisitos, los cuales encaminan la solución.

***Equipo ágil***: Conjunto de personas expertas en áreas de software y en el dominio del negocio, que trabajan orientadas hacia un mismo fin en pequeños ciclos de tiempo, llamados iteraciones. Las iteraciones se presentan varias fases: análisis, diseño, implementación y pruebas.

***Metodología Ágil***: Proceso de desarrollo de software que busca construir aplicaciones de manera rápida teniendo en cuenta el interesado para la mayoría de pasos que el método contiene. Las metodologías ágiles enfatizan en las comunicaciones cara a cara en vez de la documentación.

***Funcionalidad***: En este dominio, se define como un conjunto de características que cumplen una tarea específica en una aplicación de software.

***Iteración***: Se puede definir como la implementación de una funcionalidad en una unidad de tiempo.

***Prueba***: Procedimiento para validar una funcionalidad de la aplicación de software. Existen varios tipos de prueba, entre ellas por unidad, por componente, de exploración y de regresión.

***X-Unit***: Entorno utilizado para automatizar las pruebas unitarias. Actualmente se vienen desarrollando *frameworks* para su implementación.

***Bug Tracking System:*** Herramienta utilizada para el seguimiento de *bugs* manejada por los equipos de desarrollo para conocer qué fallos se deben corregir y cuáles no.

La filosofía de calidad toma forma en las empresas de desarrollo de software cuando todos sus miembros son probadores, pues desaparece la fase de pruebas, que forma parte de los métodos tradicionales. Es importante dejar en claro que la codificación y las pruebas constituyen un solo proceso integrado y, por ello, el tiempo empleado en codificación puede ser el mismo a emplear en las pruebas de esa aplicación de software. Es por ello que la experiencia de los probadores se hace vital a la hora de definir requisitos del cliente y su estimación de codificación y pruebas, porque el nombre ágil no significa entregar productos rápidamente con la funcionalidad solicitada, sino que, además, deben ser confiables. El diseño de pruebas, como tal, y el uso de las herramientas adecuadas se vuelven vitales para asegurar el principio de calidad. La planeación y el diseño de pruebas es importante a la hora de distribuir tiempo en *testing* particular, arreglar *bugs* y realizar otro tipo de pruebas como pruebas de aceptación, pruebas de regresión, pruebas por unidad, pruebas por componente y pruebas de exploración [5].

El factor de éxito del *testing* ágil es saber automatizar las pruebas de regresión, las cuales aseguran calidad y precisión sobre todo el comportamiento del sistema. Además, se deberían automatizar las pruebas de aceptación, que aseguran al cliente la funcionalidad que requirió en un principio [5].

## IV. ESQUEMAS PRECONCEPTUALES COMO REPRESENTACIÓN GRÁFICA DEL CONOCIMIENTO

Los esquemas preconceptuales se concibieron como herramientas útiles para la interpretación del dominio del interesado en un lenguaje controlado, denominado UN-Lencep, el cual posibilita la obtención automática de diagramas UML con el objetivo de elaborar una aplicación de software [8]. Sin embargo, en esta propuesta se pretende utilizar este esquema para la interpretación de textos, similar a lo que desarrollan Jaramillo *et al.* [13].

Para interpretar el conocimiento representado en un esquema preconceptual es necesario tener claridad sobre la sintaxis empleada. Zapata *et al.* [8] describen los componentes básicos que se encuentran en un esquema preconceptual, como se ilustran en la figura 4:

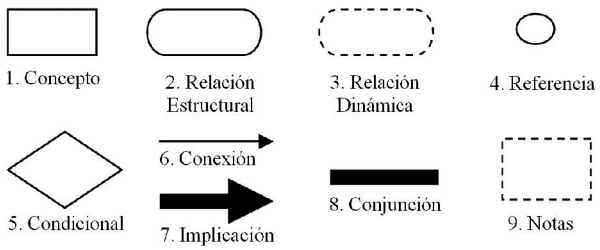


**Figura 4.** Elementos de un esquema preconceptual. Tomado de [8].

1. Concepto: Se representa por un rectángulo y representa un sustantivo o un sintagma nominal.
2. Relación Estructural: Asocia dos conceptos mediante los verbos conjugados "es" y "tiene".
3. Relación Dinámica: Asocia dos conceptos mediante verbos de acción (por ejemplo "registrar", "calcular", etc.).
4. Referencia: Se utiliza para unir dos elementos que se encuentren distantes en el diagrama.
5. Condicional: Se utiliza para definir una relación, donde se evalúa una condición que puede tener uno o dos resultados.
6. Conexión: Es una flecha que indica el sentido en que se debe leer una relación en el esquema.
7. Implicación: Es una flecha más gruesa que se utiliza para representar una relación causa-efecto.
8. Conjunción: Es una línea gruesa que aplica cuando es necesario que se cumplan más de dos condiciones para la realización de una relación dinámica.
9. Notas: Permiten especificar los valores que se pueden asignar a los conceptos.

Asimismo, se deben tener presentes las restricciones en la construcción de esquemas preconceptuales, ya que ayudan a entender mejor la manera como se realizó el discurso perteneciente a un dominio. De acuerdo Zapata y Arango [14] se deberían considerar las siguientes restricciones:

1. Los conceptos sólo pueden aparecer una vez en el esquema preconceptual y se deben definir en singular
2. En la definición de relaciones dinámicas, se debe evitar el uso de verbos generales que pueden ser muy ambiguos para el dominio, tales como: gestionar, administrar, controlar, etc.
3. En una relación dinámica el concepto inicial debe ser un actor.
4. Se deberían utilizar las funciones DIA(), MES(), AÑO() y la variable FECHA_ACTUAL cuando se requieran utilizar fechas en los condicionales.
5. En las relaciones dinámicas, se debe evitar el uso de verbos como pertenece, proviene, relaciona o asocia, ya que podrían enmascarar relaciones estructurales con relaciones dinámicas.
6. Las relaciones de implicación se deben interpretar como una actividad que se debe ejecutar para que la siguiente se pueda llevar a cabo.
7. Se debe tener cuidado con la dirección de las flechas cuando se utilizan las referencias.
8. Las conjunciones no se deben utilizar sobre relaciones estructurales.
9. Las notas se colocan únicamente para relaciones estructurales.

La simplicidad de la notación de los componentes, en un esquema preconceptual, facilita la comprensión e interpretación del interesado sobre el dominio representado. Sin embargo, para dominios extensos, es preferible cuidar la disposición que el analista realice de los elementos en el esquema, para una fácil lectura del modelo.

Uno de los beneficios de los esquemas preconceptuales, computacionalmente hablando, es que permiten, desde etapas tempranas de un desarrollo de software, la verificación del modelo contra el dominio descrito por el interesado, asegurando que se construye lo que se requiere y se garantiza la calidad que tendrá el software.

Adicionalmente, la construcción de estos esquemas reduce el tiempo en las etapas de análisis y diseño, pues tienen la ventaja de generar los diagramas de clases, comunicación y máquina de estado de UML.

## V. ESQUEMA PRECONCEPTUAL DE PRUEBAS ÁGILES

El *Testing* ágil es una técnica de pruebas de software orientada a las metodologías de desarrollo ágil, como XP, AUP, SCRUM y FDD, entre otras. En estas metodologías, interesados y equipo de desarrollo cooperan como un solo equipo para lograr un objetivo común [5]. Los diferentes roles que desempeña el *tester* ágil (analista–desarrollador) y la experiencia y habilidad que debe tener complementan el *testing* ágil. Por otra parte, en las actividades diarias de un *tester* ágil se destacan la interacción permanente con los miembros del equipo, la habilidad para integrar el enfoque del interesado con el del equipo de trabajo (analistas y desarrolladores), la facilidad en la identificación de necesidades y la aclaración de los enunciados de los requisitos y la prevención del código inadecuado en las funcionalidades del Software [5].

El rol de *tester* ágil no se limita a rutinas en el proceso de desarrollo y *testing*, sino que intenta ser innovador e introduce nuevas herramientas de optimización para automatizar la especificación y ejecución de pruebas, garantizando al interesado la elaboración de un software de calidad [5]. Dentro de las herramientas, se encuentran los DTS, útiles por toda la información que proporcionan sobre defectos del software (en relación con el entorno, el sistema operativo y el navegador, entre otros) y que se llegan a convertir en una base de conocimiento clave para la toma de decisiones sobre el software [5].

Existe, también, un abanico de pruebas, descritas en los denominados cuadrantes del *agile testing* [5] donde se esquematizan los tipos de pruebas y el grado de automatización. Estos cuatro cuadrantes (Q) tienen el propósito de ayudar al probador a comprender las técnicas de *testing* y el momento en que se deben aplicar.

Las pruebas del primer cuadrante (Q1) se orientan a las pruebas de tipo tecnológico que apoyan al equipo, como las pruebas unitarias que verifican pequeñas aplicaciones de software (métodos y algoritmos de una clase) y que se pueden automatizar empleando herramientas como *x-Unit*.

En Q2 se agrupan pruebas orientadas al negocio que apoyan al equipo, como prototipos, educción de requisitos, simulaciones, ejemplos, pruebas funcionales y *testing* por pares, entre otros.

En el Q3 se encuentran las pruebas del negocio que critican el producto, como las pruebas de exploración, pruebas de usabilidad y pruebas de aceptación.

Finalmente, en Q4 se encuentran las pruebas de tipo tecnológico que critican el producto y se orientan a la utilización de herramientas para verificar requisitos no funcionales como: pruebas de seguridad, de carga, de desempeño, pruebas de *stress* y migración de datos, entre otras.

Sin embargo, las técnicas de *testing* ágil y los valores de los *testers* no son suficientes si no se dan los cambios necesarios en la cultura en la organizacional, sobre todo en la definición de políticas de comunicación, la interrelación de las personas en el equipo, la toma de decisiones y la política de calidad. Estos cambios se deben orientar hacia la satisfacción del interesado, en la medida en que el software cumple con los requisitos, y no en entregar un producto en el menor tiempo posible. Por esto, uno de los retos a los que se ven enfrentados los equipos de aseguramiento de la calidad de una aplicación de software se relaciona con superar las dificultades para adaptarse a los nuevos roles y actividades y adoptar un cambio en el paradigma de la calidad, de acuerdo con el pensamiento ágil [5].

Lo anterior es la síntesis del *testing* ágil y el punto de partida de la representación de este discurso en UN-Lencep, que conlleva el esquema preconceptual de la figura 5. Se proponen, entonces, las siguientes relaciones estructurales del dominio, expresadas en UN-Lencep:

Equipo TIENE Proyecto
Equipo TIENE Tester
Equipo TIENE Analista
Equipo TIENE Desarrollador
Equipo TIENE Bug_Tracking_System
Proyecto TIENE Metodología
Proyecto TIENE Interesado
Proyecto TIENE Documentación
Proyecto TIENE Aplicacion_de_SW
Metodología TIENE Nombre (puede ser SCRUM, TDD, XP, AUP, ASD)
Documentación TIENE Requisito
Documentación TIENE Historia_de_usuario
Requisito TIENE ID
Requisito TIENE Descripción
Historia_de_usuario TIENE Descripción
Analista ES Tester
Desarrollador ES Tester
Bug_Tracking_System ES Herramienta
x-Unit ES Herramienta
Aplicación_de_SW TIENE Funcionalidad
Funcionalidad TIENE Código
Persona TIENE Nombre_completo
Persona TIENE ID
Tester ES Persona
Tester TIENE Habilidad
Tester TIENE Experiencia
Prueba TIENE Script
Prueba TIENE Resultado
Prueba TIENE Fecha
Prueba TIENE Tipo
Resultado TIENE Valoración
Resultado TIENE BUG
BUG TIENE ID

Las relaciones dinámicas se pueden definir a partir de los procesos que se describen en relación con el *testing* ágil, y son:

Interesado ESCRIBE Historia_de_usuario
Analista DEFINE Requisito
Tester CONSULTA Bug_Traking_System
Tester INGRESA Bug
Desarrollador CORRIGE BUG
Tester ELIGE BUG
Tester AUTOMATIZA Prueba
Tester UTILIZA x-Unit
*Interesado EJECUTA Prueba*

Las restricciones permiten establecer relaciones de causalidad entre los conceptos, y son:

- Si interesado ESCRIBE Historia_de_usuario, entonces analista DEFINE requisito.
- Si desarrollador CODIFICA funcionalidad, entonces tester UTILIZA x-Unit

- Si tester UTILIZA x-Unit, entonces tester AUTOMATIZA prueba.
- Si tester AUTOMATIZA Prueba, entonces desarrollador ESCRIBE script
- Si desarrollador ESCRIBE script, tester EJECUTA prueba
- Si Tester INGRESA BUG, entonces tester ACTUALIZA Condicion_del_Bug
- Si desarrollador elige BUG, entonces desarrollador ACTUALIZA Condicion_del_Bug
- Si desarrollador elige BUG, entonces desarrollador CORRIGE BUG
- Si desarrollador CORRIGE BUG, entonces desarrollador ACTUALIZA Condicion_del_Bug
- Si Bug_Tracking_System.BUG.Condicion_del_Bug es igual a "NO_solucionado", entonces desarrollador elige BUG

*Si interesado EVALUA funcionalidad y Prueba.tipo es igual a "aceptación" o "exploración", entonces interesado ejecuta prueba.*

En la Figura 5 se presenta el esquema preconceptual de *Agile Testing* donde se abarcaron los conceptos claves de esta técnica. Una forma de validar la utilidad del esquema preconceptual construido consiste en verificar con un caso de estudio la completitud de los conceptos, relaciones y restricciones que se establecieron en el esquema. Así, el siguiente ejemplo se tomó de Anderson y Francis [15]: "El desarrollador cambia el estado a <<se requiere más información>> e incluye una pregunta o comentario para quien reporta el *bug*. Este estado es una alerta para que, quien reporta el *bug*, suministre la información necesaria o una demostración del problema. Después de actualizar la información en el campo <<notas>>, el estado se coloca en <<verificado>> para que el desarrollador pueda continuar trabajando en el *bug*". Si este ejemplo se supone perteneciente a un entorno de desarrollo ágil, la labor de *testing* ágil asociada con este desarrollo podría tener los siguientes valores, en relación con el esquema preconceptual:

1. El desarrollador es un *tester* en *testing* ágil. En el ejemplo, el desarrollador cumple este papel y el interesado es "quien reporta el *bug*".
2. El "estado del *bug"*, en el ejemplo, se puede asociar con la "condición del *bug*". Igualmente, el "comentario o pregunta" en el ejemplo se asocia con la "descripción del *bug*" en el esquema preconceptual.
3. El campo "notas" del ejemplo, se puede asociar con la "descripción de la historia_de_usuario" en el esquema preconceptual.
4. Los estados "se requiere más información" y "verificado" del ejemplo se pueden asociar con la condición "en proceso".
5. El "cambio o colocación de un valor en el estado" del ejemplo se asocia con la "actualización de la condición del *bug*" en el esquema preconceptual.

Nótese que el esquema preconceptual puede contribuir a la automatización del proceso de *testing* ágil al verificar que se requiere alguna información adicional que el interesado aún no suministra, por ejemplo, el *bug* lo obtuvo el *tester* al ejecutar algún tipo de prueba ¿Cuál era ese tipo de prueba? Además, ¿El *bug* tiene algún código que lo pueda identificar? ¿La historia_de_usuario tiene un ID? Así, se puede notar que, además de poder representar la información incluida en el ejemplo, se puede complementar con preguntas surgidas a partir de la información requerida en el esquema preconceptual.

## VI. CONCLUSIONES Y TRABAJO FUTURO

El *testing* ágil es una técnica de pruebas que se viene consolidando como propuesta de complemento a las metodologías ágiles. En ella, todos los miembros del equipo de desarrollo pueden actuar como *testers* y se realizan actividades de prueba sobre cada una de las fases del desarrollo ágil de una aplicación de software. Pese a su utilidad, la implementación de este tipo de técnicas aún presenta problemas de comunicación al interior del equipo de desarrollo y automatización del proceso.

El esquema preconceptual de *testing* ágil que se propone en este artículo es un marco de referencia para comprender la técnica utilizada por equipos ágiles en la construcción y elaboración de pruebas en aplicaciones de software y se constituye en una síntesis de los conceptos y sus relaciones que deben tener presentes los interesados y los equipos de desarrollo al adoptar metodologías ágiles en la ingeniería del software. Esta representación ayuda a mejorar la comprensión del *testing* ágil, al igual que sirve de punto de partida para la automatización de ésta práctica de las pruebas de software. Mediante un caso de estudio, se probó que los conceptos del *testing* ágil se pueden representar con el esquema planteado y, además, se pueden generar preguntas de completitud alrededor de la información suministrada por el interesado y recabada por los miembros del equipo de desarrollo, con miras a una futura automatización de la técnica.

Se propone, como trabajo futuro, la construcción de los tres diagramas básicos de UML a partir del esquema preconceptual del *testing* ágil: el modelo estático mediante la construcción del diagrama de clases, y los modelos dinámico y de comportamiento mediante la construcción de los diagramas de comunicación y máquina de estados respectivamente. Además, se pueden generar otros diagramas, producto de la evolución que viene teniendo la teoría sobre esquemas preconceptuales.

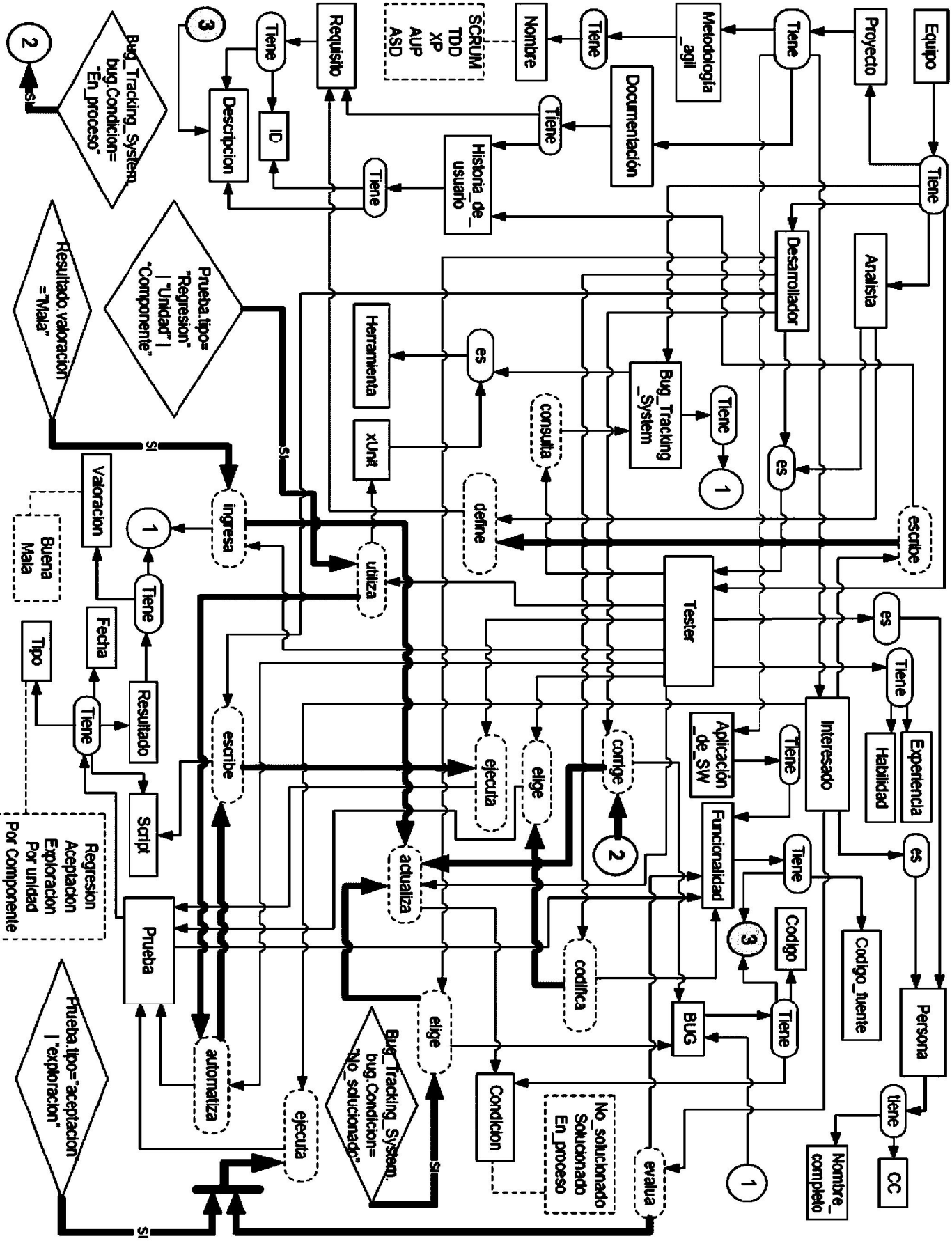


Figura 5. Esquema preconceptual sobre el dominio del *testing* ágil. Elaboración propia de los autores.

Asimismo, en cuanto a la temática representada, *testing* ágil, se propone realizar una recopilación de aplicaciones de software construidas y verificadas usando esta técnica, para evidenciar los casos de éxito, materializados en un producto terminado, que puedan servir como retroalimentación, validación y complemento del esquema preconceptual propuesto.

## REFERENCIAS


[1] Manifesto for Agile Software Development, http:// agilemanifesto.org/

[2] N. Van Schooenderwoert and R. Morsicato, 2004. Taming the embedded tiger - agile test techniques for embedded software, Agile Development Conference, Salt Lake City, pp 120-126.

[3] M. Pulejo, 2006. How not to do agile testing, Agile Conference, Minneapolis, pp. 305-314.

[4] D. Talby, A. Keren, O. Hazzan, Y., and Dubinsky, 2006. Agile software testing in a large-scale project, IEEE Software, Vol. 23, No. 4, pp. 30-37.

[5] L. Crispin and J. Gregory, v2009. Agile testing: a practical guide for testers and agile teams, Ed. Addison-Wesley Professional, Boston.

[6] S. Reid, 2009. Are all pigs equal—or are some more equal than others?, Testing Experience, No. 7, pp. 6-13.

[7] G. Lakemeyer and B. Nebel, 1994. Foundations of knowledge representation and reasoning, Ed. Springer Berlin, Heidelberg, Ch 1, pp. 1-12.

[8] C. Zapata, A Gelbukh y F. Arango, 2006. UN–Lencep: Obtención automática de diagramas UML a partir de un lenguaje controlado. 3er Taller de Tecnologías del Lenguaje Humano, Encuentro Nacional de Ciencias de la Computación, San Luis Potosí.

[9] S. Shaye, 2008. Transitioning a team to agile test methods, Agile Conference 2008, Toronto, pp 470-477.

[10] J. Novak and A. Cañas, 2008. The theory underlying concept maps and how to construct and use them, Florida Institute for Human and Machine Cognition (IHMC), Technical Report IHMC CmapTools 2006-01 Rev 01.

[11] T. Buzan, 2002. How to mind map, Ed. Thorson, London, pp 23-34.

[12] J.Geller, 1991. Propositional representation for graphical knowledge, International Journal of Man-Machine Studies, Vol. 34, Issue 1, pp. 97-131.

[13] A. Jaramillo, C. Zapata y F. Arango, 2007. Una propuesta para la asistencia al proceso de interpretación de textos utilizando técnicas de procesamiento del lenguaje natural e ingeniería de software. Revista Avances en Sistemas e Informática, Vol.4 No. 3.

[14] C. Zapata y F. Arango, 2007. Construcción Automática de Esquemas preconceptuales a partir del lenguaje natural. Informe Final del Proyecto de Investigación. C.M. Zapata (Ed.), Medellín.

[15] L. Anderson and B. Francis, 2009. The bug life cycle". Crosstalk, Vol. 16, No. 9, pp. 5-8.



**Carlos Mario Zapata.** Ingeniero Civil, Especialista en Gerencia de Sistemas Informáticos, Magíster en Ingeniería de Sistemas y Doctor en Ingeniería con énfasis en Sistemas. Profesor Asociado de la Escuela de Sistemas, Universidad Nacional de Colombia, Sede Medellín. Grupo de Investigación en Lenguajes Computacionales. Areas de Interés: Ingeniería de Software, Ingeniería de Requisitos, Lingüística Computacional y Estrategias Didácticas para la Enseñanza de la Ingeniería.

**Adrian Santiago Arboleda Ciro.** Ingeniero de Sistemas e Informática de la Universidad Nacional de Colombia, Sede Medellín. Actualmente se desempeña como Investigador en el área de Ingeniería de Software y Sistemas Inteligentes y realiza estudios de Maestría en Ingeniería de Sistemas en la Universidad Nacional de Colombia, Sede Medellín.

**Carlos Eduardo Castrillón Valencia.** Administrador de Sistemas Informáticos de la Universidad Nacional de Colombia, Sede Manizales. Actualmente se desempeña como Director de Proyectos en una empresa de desarrollo de software y realiza estudios de Especialización en Sistemas en la Universidad Nacional de Colombia, Sede Medellín.

# Universidad Nacional de Colombia Sede Medellín
# Facultad de Minas

## Escuela de Ingeniería de Sistemas

**Pregrado**

- Ingeniería de Sistemas e Informática.

**Posgrado**

- Doctorado en Ingeniería-Sistemas.
- Maestría en Ingeniería de Sistemas.
- Especialización en Sistemas con énfasis en:
  - Ingeniería de Software.
  - Investigación de Operaciones.
  - Inteligencia Artificial.
- Especialización en Mercados de Energía.

**Áreas de Investigación**

- Ingeniería de Software.
- Investigación de Operaciones.
- Inteligencia Artificial.

Escuela de Ingeniería de Sistemas
Dirección Postal:
Carrera 80 No. 65 - 223 Bloque M8A
Facultad de Minas. Medellín - Colombia
Tel: (574) 4255350 Fax: (574) 4255365
Email: esistema@unalmed.edu.co
http://pisis.unalmed.edu.co/