PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 09-17

Marco Teórico para Evaluar Taxonomías de Defectos

Fernanda Grazioli **Diego Vallespir**

2009

Grazioli, Fernanda; Vallespir, Diego
Marco teórico para evaluar taxonomías de defectos
ISSN 0797-6410
Reporte Técnico RT 09-17
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, 2009

Marco Teórico para Evaluar Taxonomías de Defectos

Fernanda Grazioli
Grupo de Ingeniería de Software
Instituto de Computación
fgrazioli@adinet.com.uy

Diego Vallespir
Grupo de Ingeniería de Software
Instituto de Computación
dvallesp@fing.edu.uy

9 de noviembre de 2009

Abstract

En este reporte se presenta un estudio y comparación de distintas taxonomías de defectos de software. Estas han sido propuestas tanto en la academia como en la industria y las que aquí se analizan son las más relevantes y conocidas. Para lograr un análisis formal desarrollamos un Marco de Comparación de Taxonomías de Defectos. Este marco original surge de mejorar y extender una Meta-Taxonomía ya existente desarrollada por Freimut.

El reporte analiza la propuesta de Freimut, presenta las características más relevantes de las taxonomías estudiadas, introduce el Marco de Comparación de Taxonomías que proponemos y muestra los resultados obtenidos al comparar las taxonomías.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Hay distintas razones para utilizar taxonomías de defectos. En *desarrollo de software* las mismas pueden ser usadas para conocer qué tipos de defectos se inyectan generalmente en una organización, en un proyecto o a nivel individual, mejorando tanto la actividad de verificación como el proceso de desarrollo de software. Las taxonomías de defectos también han sido utilizadas de distintas formas en *ingeniería de software empírica* (ESE).

Conocer qué tipos de defectos son inyectados normalmente permite buscarlos de una manera particular. Por lo tanto, la actividad de verificación puede consumir menos tiempo y encontrar más defectos. Por ejemplo, si el 90 % de los defectos se encuentran en la especificación de requerimientos, es posible verificar con mayor dedicación dicha especificación y reducir el tiempo utilizado en verificar otros productos de software. En otras palabras, orientar la verificación teniendo en cuenta el conocimiento de los defectos que se inyectan normalmente.

Además, la clasificación de defectos da una idea de las fases, actividades, disciplinas, etc. del proceso de desarrollo donde la mayoría de los defectos son inyectados. Con esta información, el proceso de desarrollo de software puede ser mejorado, logrando la reducción de la cantidad de defectos inyectados durante el desarrollo.

Por ejemplo, en ESE las taxonomías de defectos han sido utilizadas para estudiar la efectividad que tienen distintas técnicas de verificación en distintos tipos de defectos. Si se conoce esto, puede ser posible optimizar la combinación de diferentes técnicas de verificación para maximizar el número de defectos encontrados.

No sólo estos motivos, sino otros que no se exponen aquí, muestran la importancia de las taxonomías de defectos en el desarrollo de software y en ESE. Lamentablemente, no existe una taxonomía utilizada universalmente, ni en desarrollo de software ni en ESE. Esta situación causa muchos problemas, por ejemplo, es muy difícil o incluso imposible comparar resultados entre investigadores.

Existe una gran variedad de taxonomías en la industria y en la literatura. Es importante compararlas desde distintos puntos de vista, tratando de identificar las fortalezas y debilidades de cada una.

Este trabajo comienza presentando el concepto de meta taxonomía, donde se definen elementos que pueden componer una taxonomía. La meta taxonomía presentada está basada en una propuesta de Freimut. El material existente en la literatura sobre este tema es escaso. Se describen los conceptos de atributo y estructura, se presentan los atributos candidatos a ser registrados de un defecto, las distintas estructuras existentes que puede tener una taxonomía, y las propiedades deseables de una taxonomía. Se presenta una guía para incorporar una taxonomía a un proceso, una posible clasificación de taxonomías y se realiza una crítica a la propuesta de Freimut.

El trabajo continúa con la presentación de un conjunto amplio de taxonomías existentes, utilizadas en la industria y en la academia. Las mismas son la taxonomía de Hewlett-Packard, la taxonomía de Kaner, Falk y Nguyen, la taxonomía de Binder, la taxonomía propuesta por *IEEE Standard Classification for Software Anomalies*, la taxonomía propuesta por IBM y la taxonomía de Beizer.

Se crea un marco de comparación teórico original, que permite la evaluación y comparación de cualquier taxonomía de defectos de software. El marco está compuesto por dos vistas: Atributos y Propiedades. La primer vista es útil para evaluar las características de defectos que son consideradas para una taxonomía dada. La segunda vista es útil para evaluar las propiedades deseables que cada taxonomía debería tener. Se evalúan y comparan las seis taxonomías nombradas anteriormente contra el marco, presentando los resultados y conclusiones extraídas.

El reporte se organiza de la siguiente manera. La sección 2 presenta el concepto de meta taxonomía. Seis taxonomías son presentadas en la sección 3. El marco de comparación y la evaluación de las taxonomías contra el mismo se presenta en la sección 4. Finalmente la sección 5 presenta las conclusiones y el trabajo a futuro.

2. Meta Taxonomía

Una meta taxonomía define elementos que pueden componer una taxonomía, permitiendo la construcción de una taxonomía específica a partir de ella. En particular, la meta taxonomía de Freimut define atributos y propiedades. Los atributos permiten registrar las características de los defectos. Las propiedades describen propiedades deseables de una taxonomía. Una de las propiedades más relevantes es la propia estructura de la taxonomía.

En la literatura sólo encontramos la propuesta de meta taxonomía de Freimut [4], por lo que esta sección está basada en dicha propuesta.

En la sección 2.1 se define el concepto de atributo y se presentan los atributos candidatos a ser registrados de

un defecto. El concepto de estructura se define en la sección 2.2, junto a la descripción de las posibles estructuras que puede tener una taxonomía. Se presenta el concepto de propiedad y se describen las propiedades deseables de una taxonomía en la sección 2.3. En la sección 2.4 se puede observar una guía de cómo incorporar una taxonomía a un proceso. En la sección 2.5 se presenta una posible clasificación de taxonomías. Finalmente en la sección 2.6 se presenta una breve crítica al planteo propuesto por Freimut.

2.1. Atributos

Un atributo define un aspecto de un defecto. Una taxonomía está compuesta por atributos. Cada atributo tiene un conjunto de valores posibles. Cada valor representa una característica de un defecto que debe ser registrada al momento de la clasificación del mismo. Los atributos que componen una taxonomía deben ser atributos que sean relevantes para un análisis posterior. A continuación se listan los atributos propuestos por Freimut.

Ubicación Se refiere al lugar en el que se encuentra el defecto. El nivel de detalle en la especificación de esta caracteristica puede variar, ya sea indicando solamente la entidad de alto nivel del sistema en la que el defecto se ubica (Requerimientos, Diseño, Código, Documentación) o indicando además el nombre del producto en el que se encuentra.

Tiempo Se refiere a indicar la fase del proyecto en la que se produce la inyección, detección y/o corrección del defecto.

Síntoma Se refiere a lo que se observa cuando se provoca la falla. También puede ser una descripción de la actividad que provoca la falla.

Resultado final Se refiere a describir la falla causada por el defecto. Este atributo suele contener valores como performance, usabilidad, etc.

Mecanismo Se refiere a cómo se inyectó, detectó y/o corrigió el defecto. Es decir, indicar la actividad que se estaba realizando durante la inyección del defecto, la actividad que logró detectarlo o los pasos realizados para corregir el mismo.

Causa (Error) Se refiere al error humano que causó la inyección del defecto. Los valores que puede tomar este atributo serían: educación, comunicación, mal uso de herramientas, etc. Otro enfoque que se le puede dar a este atributo, es capturar distintos tipos de causas, como por ejemplo: causas humanas (falta de conocimiento, problemas de comunicación), causas del proyecto (falta de tiempo, errores de gestión), entre otras.

Gravedad Se refiere a qué tan grave es el defecto o la falla.

Costo Se refiere al tiempo o esfuerzo dedicado a encontrar o corregir un defecto. Generalmente, esta información se refiere a un nivel que representa dicho costo (mediante una escala, como puede ser: alto, medio, bajo), pero puede capturar el valor en las unidades de tiempo correspondientes.

En las definiciones propuestas por Freimut, la diferencia entre el atributo Síntoma y Resultado final no es clara. Nuestra interpretación es la siguiente. Síntoma se refiere a lo que se observa, tal vez sin saber qué es lo que realmente está pasando cuando la falla ocurre. Por otro lado, resultado final se refiere a lo que está pasando realmente cuando la falla ocurre. Por ejemplo, un síntoma de una falla puede ser un mensaje de error. Una vez que el defecto (que causa dicha falla) es detectado, se puede saber el alcance real de los efectos de la falla (resultado final). Por ejemplo, la base de datos está corrupta y un mensaje de error se despliega (el mismo mensaje que en el atributo síntoma).

2.2. Estructura

El concepto de estructura refiere a la relación existente entre los componentes de una taxonomía. Se pueden definir varios tipos de estructuras, según la relación existente entre los atributos definidos en una taxonomía o según la relación existente entre los valores posibles que puede tomar un atributo.

Una posible estructura es la jerárquica. Esto significa que existe una relación de jerarquía entre los valores posibles que puede tomar un atributo. Por ejemplo, un atributo correspondiente al tipo de defecto se puede dividir en varios subtipos y así sucesivamente. Expresado de forma genérica, los valores de un atributo en un nivel, son refinados por valores de atributo del siguiente nivel de la jerarquía. Un ejemplo de taxonomía que posee esta estructura es la taxonomía de Beizer [1].

Existe también la estructura arborescente. En dicha estructura basada en árboles, los atributos no son independientes entre sí. Es decir, la elección de un valor en un atributo, influye en los valores posibles de los siguientes atributos a clasificar.

Otra estructura posible es la ortogonal. En esta estructura los atributos son ortogonales, es decir, los valores de cada atributo son asignados independientemente de los demás atributos que componen la taxonomía. Un ejemplo de taxonomía ortogonal es la taxonomía propuesta por IBM (ODC) [3].

La estructura semi-ortogonal es aquella en la que existe una relación de semi-ortogonalidad entre los atributos de una taxonomía. Esto implica que entre algunos atributos no exista una dependencia, pero entre otros, la elección de un valor en un atributo influye en los valores posibles de otro atributo. Un ejemplo de taxonomía semi-ortogonal es la taxonomía propuesta por HP [5].

Finalmente, una taxonomía puede tener una estructura simple. En este caso, la taxonomía consiste en un único atributo donde no existen relaciones de ningún tipo entre los valores posibles que puede tomar.

2.3. Propiedades

Las propiedades son cualidades y restricciones que poseen los atributos que componen una taxonomía. Al momento de clasificar utilizando una taxonomía, si la misma no está correctamente definida, pueden surgir problemas para lograr que la información registrada durante la clasificación sea correcta, confiable y de buena calidad. Todos estos factores afectan negativamente los análisis futuros a realizar sobre la información registrada, ya que sus resultados serían cuestionables, dando lugar a dudas la validez de los mismos.

A continuación se listan algunos ejemplos de problemas que pueden surgir al tener una taxonomía incorrectamente definida: el conjunto de posibles valores que puede tomar un atributo es incompleto, o ambiguo, la definición de los atributos es confusa o incompleta, no existen ejemplos de cómo clasificar un defecto, entre otros. Para evitar estos problemas, es deseable que las taxonomías cumplan con las propiedades que se describen a continuación.

Valores de atributo mutuo-excluyentes Al momento de elegir un valor de un atributo, sólo un valor debe ser apropiado. No puede suceder que dos o más valores de un mismo atributo apliquen correctamente para un defecto, ya que esto resultaría en datos no confiables, inconsistentes.

Atributos ortogonales Como fue explicado anteriormente, la ortogonalidad refiere a que los valores de cada atributo puedan ser asignados independientemente de los demás atributos. De esta manera, se asegura que características diferentes de un defecto sean capturadas en atributos diferentes.

Completitud en valores de atributos El conjunto de valores debe ser completo para que, de esta forma, todos los defectos tengan un valor apropiado a ser asignado para cada atributo. Si el conjunto no es completo, un usuario al momento de clasificar un defecto bajo un atributo puede decidir no asignar un valor para dicho atributo o elegir el valor posible más cercano, lo cual resulta en información incompleta o inconsistente. Para prevenir este problema, es posible incluir en el conjunto de valores posibles de un atributo un valor llamado “Otro” y así, de ser necesario, en el futuro se puede ampliar la taxonomía con el valor adecuado faltante.

Menor cantidad posible de valores de atributos El conjunto de valores posibles para un atributo debe ser lo más pequeño posible. Es importante que si una taxonomía va a ser utilizada frecuentemente por desarrolladores, sea lo más fácil posible de aplicar y que no requiera demasiado tiempo en clasificar un defecto. El tener un conjunto pequeño de valores, no sólo hace más simple el proceso de clasificación, sino que lo hace menos propenso a errores humanos. Por otro lado, si la taxonomía a aplicar está diseñada para un futuro análisis profundo, es posible que se necesite un conjunto con una mayor cantidad de valores posibles para lograr una precisión adecuada en los resultados.

Descripción de valores de atributos Es fundamental que cada uno de los valores posibles que puede tomar un atributo esté claramente definido y detallado con ejemplos de defectos que clasifican con cada valor. El objetivo de esta propiedad es evitar las posibles confusiones y malas interpretaciones, ya que las mismas pueden resultar en datos inconsistentes y no confiables.

Las cinco propiedades explicadas anteriormente, son propiedades deseables que debería tener una taxonomía. Para lograr un mejor uso de una taxonomía, hay una propiedad adicional que se puede agregar a la lista anterior: consistencia en la aplicación. Esta propiedad se refiere a mantener una consistencia en las taxonomías utilizadas durante distintas etapas de un proceso y en distintos productos o proyectos. Esta propiedad plantea el uso de una misma taxonomía en todas las etapas de desarrollo. De esta forma se evita que un mismo usuario tenga que clasificar con más de una taxonomía, pudiendo confundirse al aplicar las mismas. El hecho de utilizar una misma taxonomía en distintos proyectos es lo que permite realizar análisis comparativos y además de brindar la posibilidad de descubrir patrones de defectos independientes del proyecto o proceso.

2.4. Incorporación

Al momento de incorporar una taxonomía a una empresa, se puede optar por utilizar una taxonomía ya existente y de ser necesaria adaptarla a un contexto, o desarrollar una nueva taxonomía basada en las necesidades y objetivos de la empresa. A continuación se describen ideas de cómo llevar a cabo estos dos caminos posibles.

Para una empresa que simplemente quiere tener una idea de los tipos de defectos encontrados o que no quiere invertir en desarrollar una taxonomía propia, utilizar una taxonomía existente es su mejor opción. La taxonomía que es generalmente utilizada y/o adaptada según las necesidades es ODC.

Sin embargo, cuando se decide crear una nueva taxonomía, hay que definir los atributos a incluir y sus valores apropiados, según el análisis que posteriormente se desea realizar con esos datos y las metas que se quieran cumplir.

En [7] se presenta una guía de pasos para organizaciones que desean reutilizar la taxonomía propuesta por la IEEE, cómo adaptarla a su realidad e incorporarla a la organización. Estos mismos pasos pueden servir como guía para el desarrollo de una nueva taxonomía de defectos.

- Elegir los atributos de manera que cumplan las propiedades del standard, basándose en el objetivo del análisis para realizar la elección.
- Para cada atributo, determinar su conjunto de valores posibles. Se recomienda comenzar con un conjunto base, y probar clasificar el atributo en cuestión para una muestra de defectos. De esta manera se chequea que los valores del conjunto base tengan sentido, si no existe más de una opción válida para un mismo defecto y la completitud del conjunto. El conjunto de valores va a ir creciendo de ser necesario.
- Documentar los atributos, describiendo claramente cada uno de ellos utilizando el vocabulario típico de la empresa, de forma que los usuarios que utilicen la taxonomía comprendan claramente y sin ambigüedades. En este sentido también se debe documentar cada valor posible que puede tomar un atributo.
- Especificar cuándo y cómo dicha información va a ser recolectada, es decir, explicar el sistema de registro y seguimiento de defectos.
- Planificar los análisis que se desean realizar y en qué momento se deben ejecutar los mismos.
- Brindar capacitación adecuada para todos los usuarios de la taxonomía.

En todos los pasos es bueno ir recordando y verificando el cumplimiento de las propiedades deseables para una taxonomía.

2.5. Clasificación de taxonomías

En [12] se presenta una clasificación de lo que el autor llama *defect classification systems*. Estos sistemas se dividen en tres categorías: Taxonomías de defectos, *root cause analysis* y clasificación de defectos.

Las taxonomías de defectos son categorizaciones de defectos (generalmente defectos de codificación), que están basadas en los detalles de la implementación de su solución. Por ejemplo, declaración de un tipo incorrecto, alcance de variable incorrecto, o mal manejo de interrupciones. Las taxonomías de defecto sólo clasifican al defecto en una única dimensión (es decir, están compuestas por un único atributo). Un ejemplo bien claro de una taxonomía de defectos es la taxonomía de Beizer.

Una propuesta más detallada es *root cause analysis*, donde no sólo se analizan los defectos por sí solos, sino también su causa. El objetivo es identificar la raíz de estas causas y eliminarlas para prevenir la inyección de los mismos defectos en el futuro. En general, *root cause analysis* es visto como un enfoque que requiere bastante elaboración y su relación costo/beneficio no está clara. Algunas propuestas de *root cause analysis* se presentan en [10] y [9].

Las clasificaciones de defectos tienen como meta reducir los costos y mantener los beneficios de *root cause analysis*. Clasifican un defecto en múltiples dimensiones (están compuestas por varios atributos). Ejemplos de clasificaciones de defectos son las taxonomías propuestas por IBM, HP y la propuesta en el standard de la IEEE.

En este reporte, utilizaremos el término taxonomía para todos los *defect classification systems*.

2.6. Discusión

En esta sección planteamos algunas críticas a la propuesta de Freimut sobre los atributos a incorporar en una taxonomía.

Al leer su propuesta, se encontró cierta dificultad en su interpretación. Sus definiciones no son claras, resultan ambiguas en algunos casos, lo que nos indujo a la confusión en varias oportunidades.

Notamos que un mismo atributo describe distintos aspectos de un defecto. Por ejemplo, el atributo Síntoma abarca la descripción de una falla y la actividad que provoca la falla.

Bajo el atributo Mecanismo, se incluyen actividades de inyección, detección y corrección de defectos. Sin embargo, cuando propone un ejemplo, plantea que una actividad de detección son las pruebas unitarias. Esto lleva a una contradicción, ya que la prueba unitaria es en realidad una actividad que produce una falla, no detecta un defecto. Y aquí surge la ambigüedad y confusión entre Síntoma y Mecanismo. Nosotros entendemos que una actividad de detección sería aquella que permite ubicar el defecto en un producto, por ejemplo aplicar técnicas de *Debugging*, lo cual es distinto de una actividad que provoca la falla por primera vez, como puede ser la ejecución de un caso de prueba.

3. Taxonomías

Esta sección tiene como objetivo presentar un amplio conjunto de taxonomías particulares, algunas utilizadas en la industria y otras sólo académicamente, de manera observar distintos enfoques de un mismo concepto.

En la sección 3.1 se presenta la taxonomía propuesta por Hewlett-Packard. La taxonomía propuesta por Kaner, Falk y Nguyen se presenta en la sección 3.2. La taxonomía presentada por Binder se explica en la sección 3.3. En la sección 3.4 se detalla el *IEEE Standard Classification for Software Anomalies*. La taxonomía propuesta por IBM, *Orthogonal Defect Classification*, se presenta en la sección 3.5. En la sección 3.6 se describe la taxonomía presentada por Boris Beizer. Finalmente en la sección 3.7 se presenta un mapeo realizado por Freimut entre los atributos presentados en la sección 2 y algunas taxonomías.

3.1. Taxonomía de Hewlett-Packard

La taxonomía propuesta por Hewlett-Packard [5] fue desarrollada en 1986 con el objetivo de mejorar el proceso de desarrollo a través de la reducción de defectos en el tiempo. Está diseñada para ayudar a una organización en el estudio de las tendencias de defectos en productos ya culminados y utilizar esa información para la prevención de defectos en futuros proyectos.

La taxonomía puede verse claramente en la figura 1. La misma posee una estructura semi-ortogonal, y está compuesta por tres atributos a clasificar. A continuación se presenta cada uno de ellos.

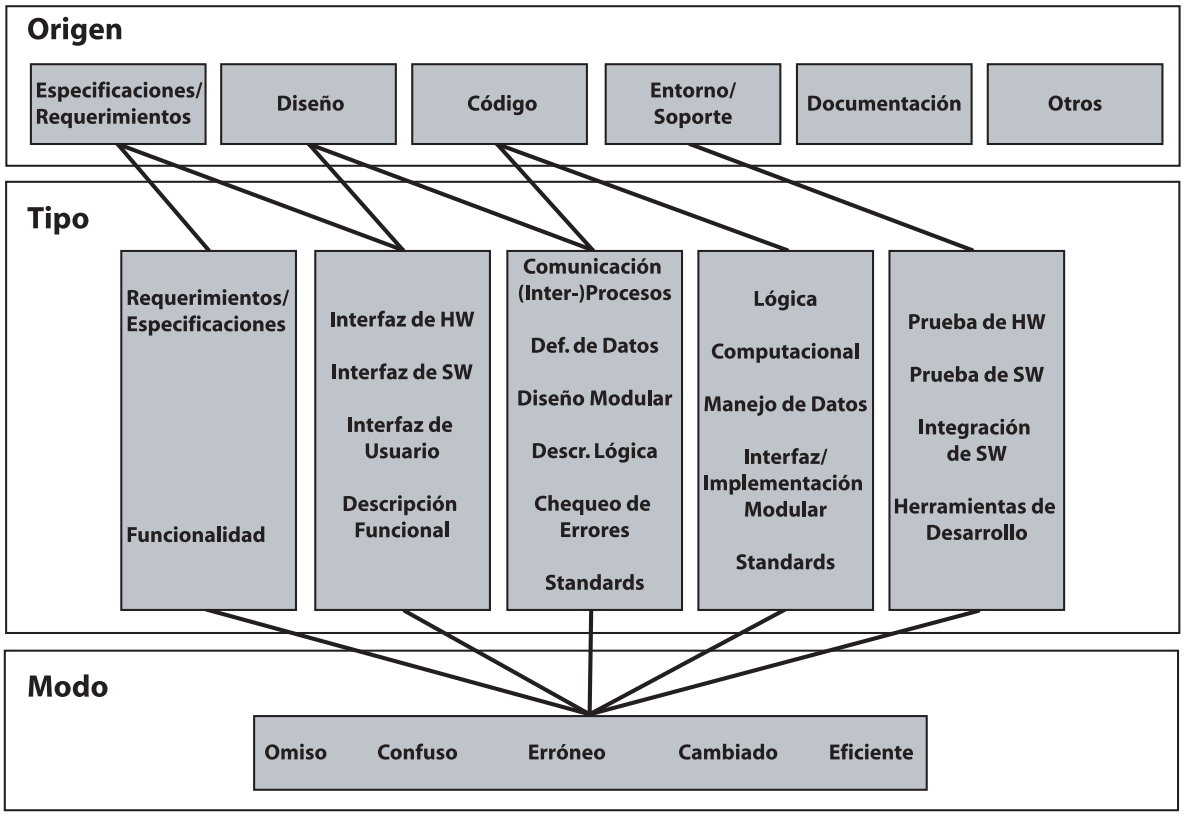

Figura 1. Taxonomía de Hewlett-Packard

3.1.1. Origen

Se refiere a la actividad del proceso donde el defecto fue inyectado. Los valores posibles para el atributo Origen son los siguientes:

Especificaciones/Requerimientos Se refiere a las tareas relacionadas con la determinación de las necesidades y de las condiciones a satisfacer para el software en cuestión.

Diseño Se refiere a la actividad que determina y especifica el funcionamiento del software en forma general, sin entrar en detalles de implementación.

Código Se refiere a la tarea de implementación, codificación.

Entorno/Soporte Son las actividades relacionadas al entorno de desarrollo y soporte técnico a los usuarios.

Documentación Es todo lo concerniente a la documentación del desarrollo del software y de la gestión del proyecto en cuestión.

Otros Se refiere a cualquier otra actividad involucrada en el desarrollo.

3.1.2. Tipo

Se refiere al área que es responsable por el defecto (sus valores dependen del origen seleccionado previamente). Los valores posibles para el atributo Tipo son los siguientes:

Requerimientos/Especificaciones Un defecto de este tipo produce que el producto no cumpla con las características deseadas.

Funcionalidad Un defecto de este tipo afecta la funcionalidad del producto.

Interfaz de HW Son defectos relacionados con la interfaz de hardware, es decir, problemas a nivel de los dispositivos utilizados para ingresar, procesar y entregar los datos.

Interfaz de SW Son defectos relacionados con la interfaz de software destinada a entregar información acerca de los procesos y herramientas de control.

Interfaz de Usuario Son defectos relacionados con la interfaz que se encarga de la interacción usuario-máquina.

Descripción Funcional Son defectos en la descripción de los requerimientos funcionales o diseño funcional del sistema.

Comunicación (Inter-)Procesos Se refiere a defectos causados por problemas de comunicación entre procesos.

Definición de Datos Son defectos relacionados con la definición de estructura de datos.

Diseño Modular Se refiere a defectos relacionados a la modularización del sistema.

Descripción Lógica Son defectos relacionados al formalismo lógico de la descripción de un diseño o código.

Chequeo de Errores Se refiere a la omisión o incorrectitud al chequear errores.

Standards Se refiere al cumplimiento de standards de diseño.

Lógica Son defectos relacionados a la lógica de la implementación.

Computacional Se refiere a defectos relacionados a errores computacionales.

Manejo de Datos Se refiere a defectos relacionados con el manejo de datos.

Interfaz/Implementación Modular Son defectos relacionados con la implementación de módulos o interfaces.

Standards Se refiere al cumplimiento de standards de implementación.

Prueba de HW Son defectos relacionados con pruebas de hardware.

Prueba de SW Son defectos relacionados con pruebas de software.

Integración de SW Se refiere a defectos en la integración de software.

Herramientas de Desarrollo Son defectos relacionados con las herramientas de desarrollo utilizadas en el proyecto.

3.1.3. Modo

Se refiere a la causa que llevó al defecto. Los valores posibles para el atributo Modo son los siguientes:

Omiso Significa que hay que agregar algo para corregir el defecto.

Confuso Significa que algo no está claro.

Erróneo Significa que el defecto se debe a un error en algo realizado.

Cambiado Se refiere a que el defecto se introdujo al cambiar otra parte para corregir otro defecto.

Eficiente Se refiere a que se puede mejorar haciéndolo de otra manera más eficiente.

3.2. Taxonomía de Kaner, Falk y Nguyen

En el libro *Testing Computer Software* [8], los autores Kaner, Falk y Nguyen presentan una taxonomía compuesta por un único atributo que es el tipo de defecto. El conjunto de valores posibles para dicho atributo está formado por 13 grandes categorías y donde cada categoría contiene un conjunto de subcategorías asociadas. Es decir, es una taxonomía con una estructura jerarquíca de dos niveles. Los valores no están totalmente descriptos en el libro, ya que existen valores para los cuales no se explica su intención. Por lo tanto, a continuación se describe cada una de las categorías y subcategorías asociadas, realizando una interpretación de las mismas.

3.2.1. Defectos de Interfaz de Usuario

Son defectos relacionados a la interfaz encargada de la interacción entre usuarios y computadoras. Las subcategorías de esta categoría son:

Funcionalidad La funcionalidad es poco manejable, difícil, confusa o imposible de realizar con la interfaz propuesta.

Comunicación La información que provee la interfaz no es suficiente, es confusa o no es exacta.

Estructura de Comandos Se refiere a que los comandos son confusos o que es fácil perderse dentro del programa.

Comandos Faltantes Se refiere a lo que está faltando o a si el programa fuerza al usuario a actuar en una manera antinatural.

Performance Se refiere mayormente a la sensación de velocidad del software en el sentido de interacción con el usuario.

Salida Se refiere a las salidas impresas en pantalla, si tienen sentido o si los gráficos son claros.

3.2.2. Manejo de Errores

Esta categoría se refiere a defectos en el manejo de errores del sistema. Las subcategorías de esta categoría son:

Prevención de Errores Se refiere a los defectos que surgen al anticipar la posibilidad de errores y proteger al sistema de ellos.

Detección de Errores Se refiere a defectos al anunciar condiciones de errores o al hacer frente a los errores detectados.

Recuperación de Errores Se refiere a defectos relacionados con rutinas de recuperación de errores que se ejecutan para volver al sistema a un estado conocido y estable.

3.2.3. Defectos Relacionados a Fronteras

Se refiere a defectos relacionados con fronteras de dominios o condiciones límite. Las subcategorías de esta categoría son:

Fronteras numéricas Se refiere a defectos en las fronteras del dominio de un valor numérico.

Frontera en condiciones Se refiere defectos a las fronteras de una condición lógica.

3.2.4. Defectos en Cálculos

Esta categoría abarca los defectos relacionados con fórmulas, operaciones, precisión de resultados y errores computacionales. Las subcategorías de esta categoría son:

Constantes no actualizadas Se refiere a la utilización de un valor constante que no se corresponde con el valor válido en la actualidad.

Orden incorrecto de operadores Se refiere a la mal interpretación de fórmulas o colocación errónea de operadores en la expresión.

Overflow* y *underflow Surgen cuando alguna operación aritmética produce un resultado fuera del rango representable.

Errores computacionales Se refiere a defectos al aplicar fórmulas a datos que no son aplicables, pérdida de precisión debido al redondeo o truncamiento.

3.2.5. Estado Inicial y Siguientes

Esta categoría abarca los defectos de inicialización y actualización de variables. Las subcategorías de esta categoría son:

Inicialización de variable Se refiere a defectos que se relacionan a la inicialización de una variable.

Inicialización de variable de control de un bucle Se refiere a defectos que se relacionan a la inicialización de una variable de control en un bucle (variable que participa en la condición del bucle).

Reinicialización Se refiere a defectos relacionados con asignaciones de valores a variables.

3.2.6. Defectos de Control de Flujo

Esta categoría se refiere a los defectos en el orden en el que se ejecutan las instrucciones. Las subcategorías de esta categoría son:

Running Amok El programa ejecuta de manera incontrolable, sin parar.

Finalización inesperada El programa finaliza su ejecución inesperadamente.

Bucles Se refiere a errores de control de flujo relacionados con bucles.

if-then-else Se refiere a errores de control de flujo relacionados con sentencias del tipo *if-then-else*.

3.2.7. Defectos de Manejo o Interpretación de Datos

Esta categoría se refiere a los defectos en el manejo y/o interpretación de datos intercambiados entre módulos o entre programas. Las subcategorías de esta categoría son:

Tipos de datos Se utiliza un dato como si fuese de un tipo, pero en realidad es de otro tipo.

Lista de parámetros Se refiere a defectos introducidos por la invocación incorrecta de un método, ya sea por utilizar parámetros fuera de orden o por la omisión de alguno de ellos.

Copias de datos desactualizados El valor de la copia del dato utilizado no se corresponde con el valor actual real del dato.

Valor erróneo de una tabla Se consulta un valor equivocado en una tabla.

Máscara incorrecta En el manejo de bits, la máscara aplicada no es la correcta.

3.2.8. Condiciones de Carrera

Se llama carrera a una competencia entre dos eventos A y B, donde cualquiera de los dos pueden ser los siguientes en ser ejecutados. Si A ocurre primero, el programa funciona correctamente. Si B ocurre antes que A, el programa falla debido a que se esperaba que A ocurriese antes que B. El programador no se percató que B podría "ganar la carrera", y B gana sólo en condiciones especiales. Las subcategorías de esta categoría son:

Asumir órden de eventos Se asume que un evento siempre termina antes que otro, cuando eso no es necesariamente así.

Asumir rango de entrada Se asume que una entrada no va a ocurrir en un intervalo específico.

Ejecución temprana Las tareas comienzan su ejecución antes que se cumplan sus tareas previas.

3.2.9. Condiciones de Carga

Esta categoría se refiere a los defectos por sobrecarga. Las subcategorías de esta categoría son:

Falta de recursos Los recursos requeridos no están disponibles.

Liberación de memoria No se libera la memoria que ya no se está utilizando.

3.2.10. Hardware

Esta categoría abarca los defectos involucrados con el hardware, como el envío incorrecto de datos a los dispositivos o el intentar utilizar recursos no disponibles. Las subcategorías de esta categoría son:

Dispositivo no está disponible No se puede acceder al dispositivo deseado o el mismo está ocupado.

Fin de archivo inesperado El dispositivo o el programa recibe datos incompletos o en mal formato.

3.2.11. Control de Versión y Fuentes

Esta categoría abarca los defectos causados por un mal manejo de versiones y código fuente. Las subcategorías de esta categoría son:

Reaparición de defectos Viejos defectos ya corregidos vuelven a aparecer por volver a una versión anterior.

Falta de correspondencia El código fuente no se corresponde con el binario.

3.2.12. Documentación

Aquí clasifica cualquier defecto relacionado con la documentación, ya sea del área de Requerimientos, Diseño, Implementación o Soporte. En la taxonomía no se detallan las subcategorías correspondientes.

3.2.13. Defectos de Testing

Se refiere a defectos en el área de verificación, como defectos al planificar, diseñar y/o implementar casos de pruebas o errores cometidos en los reportes de defectos. En la taxonomía no se detallan las subcategorías correspondientes.

3.3. Taxonomía de Robert Binder

Robert Binder, en su libro *Testing Object-Oriented Systems Models, Patterns, and Tools* [2], muestra que en la programación orientada a objetos gran parte de los defectos se deben a problemas en la aplicación de los conceptos de herencia, polimorfismo, secuencia de mensajes y transición de estados. Los defectos surgen mayormente al aplicar dichos conceptos porque los mismos se usan con frecuencia en la programación orientada a objetos, ya que forman la base de este paradigma. Además, estos conceptos son muy distintos a los conceptos básicos de la programación procedural.

A continuación se presenta una interpretación de la taxonomía propuesta por Binder para el paradigma orientado a objetos a nivel de Método, Clase y *Cluster*, donde él único atributo a clasificar el tipo de defecto. El autor aclara que aún se debe trabajar sobre la taxonomía, ya que la misma no está completa y tampoco cumple con el *IEEE Standard Classification for Software Anomalies*. Se presenta una interpretación de la taxonomía, ya que en el libro no se describe ni se explica el conjunto de valores propuestos. En el caso en que no se haya interpretado algún valor, se pondrá “No se interpretó lo planteado por el autor”.

3.3.1. Taxonomía de Defectos a Nivel de Método

La misma presenta los tipos de defecto posibles a nivel de método para las áreas de Requerimientos, Diseño e Implementación.

A continuación se describen los posibles tipos de defecto para el área de Requerimientos.

Omisión de requerimiento Los requerimientos no fueron relevados completamente.

Transformación incorrecta u omisa Hay una mala interpretación del requerimiento planteado por el cliente o falta interpretar el mismo.

A continuación se describen los posibles tipos de defecto para el área de Diseño.

Baja cohesión Los métodos de una misma clase realizan tareas variadas o que no tienen demasiada relación.

Conflicto de resposabilidades Las responsabilidades se superponen o entran en conflicto con métodos locales o de superclases.

Feature override missing No se interpretó lo planteado por el autor.

Feature delete missing No se interpretó lo planteado por el autor.

Uso excesivo de mecanismos *friend/protected* Se refiere que al momento de determinar las visibilidades, se utiliza demasiado las propiedades *friend/protected*, cuando podrían ser *public/private*.

Naked access No se interpretó lo planteado por el autor.

Método excesivamente largo Se refiere a la longitud en líneas de código y en tareas que realiza del método.

Método sin código El método está vacio, no tiene contenido.

Atributos excesivamente largos Se refiere a la longitud de los atributos.

Algoritmo incorrecto El algoritmo no realiza lo que debería hacer.

Algoritmo ineficiente El algoritmo no es eficiente, problemas de performance.

Violación de invariante Algo que no debería cambiar se modifica.

Excepción no lanzada No se lanza una excepción cuando debería ser lanzada.

Excepción no atrapada No se atrapa una excepción cuando debería ser atrapada.

A continuación se describen los posibles tipos de defecto para el área de Implementación.

Mensaje sin correspondiente Un mensaje es enviado a un objeto que no contiene el método correspondiente.

Código inalcanzable Hay código que jamás se va a ejecutar.

Violación de contrato No se cumple con el contrato del método, ya sea con la precondición, poscondición o invariante.

Mensaje enviado a un servidor de objetos equivocado El mensaje que se envía no era para el servidor de objetos al cual se envió.

Parámetros de mensaje Los parámetros del mensaje son incorrectos u se omitió alguno de ellos.

Prioridad de mensaje La prioridad del mensaje no es la correcta.

Mensaje no implementados en el servidor El mensaje no fue implementado en el servidor correspondiente.

Error de sintáxis Se refiere a cualquier error sintáctico del método.

Algoritmo ineficiente El algoritmo no está implementado de la manera más eficiente, pudiendo introducir problemas de performance o recursos.

Salida incorrecta La salida del método no es la esperada.

Precisión La precisión es incorrecta, incluye error de redondeo o truncamiento.

Persistencia incorrecta Los objetos no son salvados correctamente.

Algoritmo no finaliza El método no termina nunca de ejecutar.

Excepción faltante Falta implementar una excepción.

Excepción incorrecta La excepción lanzada no es correcta.

Excepción no atrapada Una excepción no fue atrapada cuando debió atraparse.

***Catch* incorrecto** Una excepción fue atrapada cuando no debería haberse atrapado.

Excepción excede alcance Una excepción se propaga fuera del alcance determinado.

Excepción no se lanza Una excepción no fue lanzada cuando debió lanzarse.

Estado incorrecto luego de una excepción El sistema, luego de lanzar una excepción, queda en un estado que no es en el que debería estar.

Objeto faltante Se referencia a un objeto que no está definido.

Objeto sin utilizar Se define un objeto que no es referenciado.

Inicialización faltante, constructor incorrecto Falta inicializar una variable o el contructor de la clase no está correctamente implementado.

Contrato de servidor violado No se respetó el contrado del servidor.

Unidades Se refiere al uso incorrecto de unidades, como por ejemplo confundir gramos con onzas.

Visibilidad/alcance incorrecto La visibilidad del método no fue definida correctamente.

Serialización incorrecta Se realiza una incorrecta serialización, lo cual resulta en un estado corrupto.

3.3.2. Taxonomía de Defectos a Nivel de Clase

La misma presenta los tipos de defecto posibles a nivel de clase para las áreas de Requerimientos, Diseño, Implementación y Proceso.

A continuación se describen los posibles tipos de defecto para el área de Requerimientos.

Omisión de requerimiento Los requerimientos no fueron relevados completamente.

Dominio no identificado de un objeto No se logra identificar correctamente el dominio del objeto.

Nivel de esfuerzo, límites del sistema El nivel de esfuerzo es incorrecto o los límites del sistema no están bien definidos.

Nivel de abstracción El nivel de abstracción es incorrecto o inapropiado.

Diagrama de estados El diagrama de estados no está correctamente definido.

Asociación/Multiplicidad Omisión de asociación o con multiplicidad incorrecta.

A continuación se describen los posibles tipos de defecto para el área de Diseño.

Asignación de responsabilidades La asignación de responsabilidades es redundante, es decir, el mismo método aparece en varias subclases sin estar definido en una superclase en común.

Diseño de objetos *top-level* y protocolo Estructura de clases pobre, protocolos no ortogonales, jerarquía y herencia inconsistentes.

Asociación Asociación faltante o incorrecta.

Tipos de parámetros Los tipos de parámetros especificados para una clase genérica son incorrectos.

Bucles de herencia Por ejemplo, C es superclase de A, A es superclase de B y B es superclase de C.

Incorrecta redefinición de método en subclase Una subclase incorrectamente redefine un método de su superclase.

Característica incorrectamente heredada La característica es heredada cuando no debería ser así.

Especificación incorrecta Faltan métodos o variables en la especificación.

Ciclo en jerarquía Existen ciclos en la jerarquía.

Herencia múltiple La herencia múltiple fue incorrectamente definida.

Colocación de clase La clase está ubicada en un lugar no apropiado.

Interfaz pública Se coloca una interfaz pública a una clase, pero no a través los métodos de la clase.

Comunicación inter-clase Comunicación implícita clase a clase.

Uso excesivo de mecanismos *friend/protected* Se refiere que al momento de determinar las visibilidades, se utiliza demasiado las propiedades *friend/protected*, cuando podrían ser *public/private*.

Métodos *public/private* No se utilizan los métodos *public/private*.

Objeto no utilizado Un objeto está definido pero no se referencia.

Número excesivo de métodos La cantidad de métodos de la clase es demasiado grande.

Demasiados atributos Se utiliza una cantidad muy grande de atributos.

Módulo, clase, método o atributo no utilizado Algo que está definido pero que no se utiliza/referencia.

Módulo no contiene clases El módulo definido no contiene clases, está vacio.

Método no contiene código Existe un método que no contiene código, está vacio.

Clase no contiene métodos La clase no contiene ningún método definido.

Clase sin atributos La clase no tiene ningún atributo definido.

Módulo excesivamente largo El módulo definido es demasiado grande.

Diagrama de estados El diagrama de estados no está correctamente definido.

Inconsistencia de contrato El contrato no es consistente consigo mismo o con el resto del sistema.

Concurrencia Hay una incorrecta especificación sobre la concurrencia.

Violación de invariante Algo que no debería cambiar se modifica.

A continuación se describen los posibles tipos de defecto para el área de Implementación.

Ubicación Ubicación del método incorrecta debido a sinónimos, errores de nombrado.

Constructor/Destructor Incorrecta implementación del contructor o destructor de la clase.

Parámetros Parámetros incorrectos usados en un clase genérica.

Instanciación Instanciación de una clase abstracta.

Error de sintáxis Se refiere a cualquier error sintáctico.

Asociación Se refiere a una asociación no implementada.

A continuación se describe los posibles tipos de defecto para el área de Proceso.

Documentación La documentación de la clase es inconsistente con su implementación.

Patrones/Standards No se aplican patrones de diseño o standards de codificación.

Versión Uso incorrecto del control de versionado.

3.3.3. Taxonomía de Defectos a Nivel de *Cluster*

Se llama *cluster* a un grupo relacionado de clases. La idea es no testear una clase por separado, sino un conjunto de clases con el objetivo de probar una funcionalidad más grande. Los subsistemas son un tipo de *clusters* más grandes, que se incluyen en este nivel.

Esta taxonomía presenta los tipos de defecto posibles a nivel de *cluster* para las áreas de Requerimientos, Diseño e Implementación.

A continuación se describen los posibles tipos de defecto para el área de Requerimientos.

Omisión de requerimiento Los requerimientos no fueron relevados completamente.

Nivel de abstracción El nivel de abstracción es incorrecto o inapropiado.

Diagrama de estados El diagrama de estados no está correctamente definido.

Asociación Omisión de asociación.

A continuación se describen los posibles tipos de defecto para el área de Diseño.

Exportación de estructura La implementación de la estructura de datos fue exportada incorrectamente.

Exportación de característica La exportación de una característica fue realizada incorrectamente.

Visibilidad de paquete Incorrecta visibilidad de paquete.

Métodos no locales La clase contiene métodos no locales.

Método no utilizado Hay métodos *public* que no son usados por otros objetos.

Mensaje/Objeto incoherente El mensaje no se corresponde con el objeto al que debe ser enviado.

Interfaz con el entorno La interfaz con el entorno no está correctamente definida.

Componente omiso Hay algún componente que falta en el *cluster*.

Componente inconsistente El componente no es consistente con la definición.

Baja cohesión Hay una baja cohesion entre los componentes del *cluster*.

A continuación se describen los posibles tipos de defecto para el área de Implementación.

Prioridad La prioridad definida no es la correcta.

Serialización La serialización no está implementada correctamente.

Mensaje a objeto inexistente Se envía un mensaje a un objeto que fue destruido previamente.

Recolección de basura El recolector de basura es inconsistente, no funciona adecuadamente.

Mensaje equivocado a objeto correcto Se envía un mensaje erróneo al objeto deseado.

Excepción correcta en objeto equivocado Se lanza una excepción correcta, pero la misma no debe ser lanzada por ese objeto.

Excepción equivocada en objeto correcto Se lanza una excepción que no debería lanzarse en el objeto correcto.

Pedido/Devolución de recursos Se realiza un pedido o devolución de recursos incorrectamente.

Concurrencia Se refiere a los problemas relacionados con la implementación de la concurrencia.

Performance La performance no es la esperada.

Bloqueo mutuo El sistema queda inoperante en estado de bloqueo mutuo debido a problemas de concurrencia.

3.4. IEEE Standard Classification for Software Anomalies

El *IEEE Standard Classification for Software Anomalies* [6] está pensado para organizaciones que quieren implementar o ampliar una taxonomía o para aquellas que deseen expandir y mejorar sus mecanismos de registro y seguimiento de defectos con metodologías ya probadas y analizadas.

El standard propone una taxonomía de atributos ortogonales cuya clasificación se debe realizar en cuatro etapas: Reconocimiento, Investigación, Acción y Disposición. En cada una de las etapas, se debe registrar lo encontrado/investigado/decidido/realizado, clasificar los atributos correspondientes e identificar su impacto. Si bien sus atributos son ortogonales, dentro de cada atributo existe una relación jerárquica dentro del conjunto de valores posibles, por lo que la estructura de la taxonomía es ortogonal y jerárquica.

A continuación se describe cada una de las etapas, junto con los atributos obligatorios y opcionales a clasificar en cada una de ellas.

3.4.1. Reconocimiento

Esta etapa comienza cuando se genera una falla y por lo tanto un defecto es descubierto. Esta etapa de clasificación puede ser efectuada por cualquiera que esté involucrado en el desarrollo, testing, uso o mantenimiento de software, sin importar la etapa en la que se encuentra el producto en cuestión.

Los atributos a clasificar dentro de la etapa de Reconocimiento son los siguientes:

Estado del Producto Se refiere a cuál es la usabilidad del producto en ese momento (sin realizar ningún cambio). Este atributo es opcional.

Actividad del Proyecto Se refiere a lo que se estaba realizando cuando se produjo la falla. Este atributo es obligatorio.

Fase del Proyecto Se refiere a la fase del ciclo de vida en la que se encuentra el proyecto. Este atributo es obligatorio.

Repetibilidad Se refiere a cuántas veces puede suceder la anomalía. Este atributo es opcional.

Causa Sospechada Se refiere a lo que se piensa que causó del defecto. Los valores posibles dan un aspecto de ubicación sospechada. Este atributo es opcional.

Síntoma Se refiere a cómo se manifestó la anomalía. Este atributo es obligatorio.

Los atributos relacionados al Impacto que se deben clasificar dentro de la etapa de Reconocimiento son los siguientes:

Valor para el Cliente Se refiere a qué tan importante es corregir este defecto para el cliente. Este atributo es opcional.

Misión/Seguridad Se refiere a qué tan mala es la anomalía respecto a los objetivos del proyecto o al bienestar humano. Este atributo es opcional.

Gravedad Se refiere a qué tan grave es la anomalía, a nivel de desarrollo del producto (no a nivel de cliente). Este atributo es obligatorio.

3.4.2. Investigación

Es cuando el defecto ya fue ubicado, investigado y analizado lo suficiente como para identificar su causa, proponer soluciones o indicar que el defecto no requiere acción.

Los atributos a clasificar dentro de la etapa de Investigación son los siguientes:

Causa Actual Se refiere a la causa real del defecto. Los valores dan un aspecto de ubicación actual. Este atributo es obligatorio.

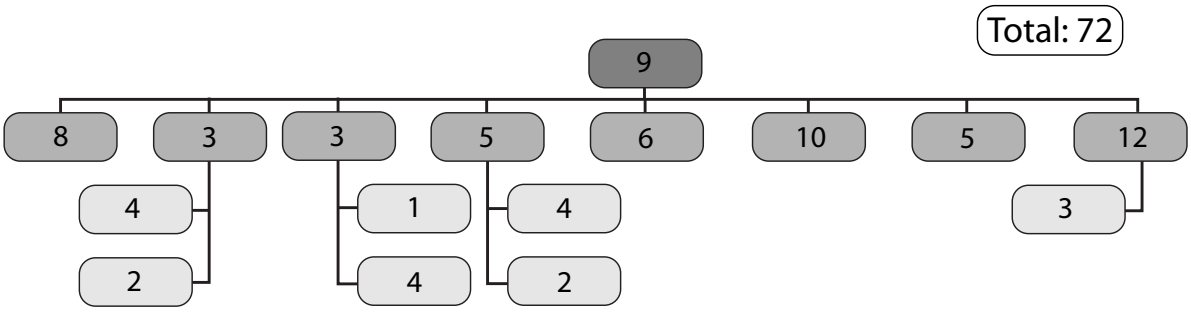

Figura 2. Jerarquía del atributo Tipo en IEEE

Fuente Se refiere a dónde se originó el defecto. Este atributo es obligatorio.

Tipo Se refiere a qué tipo de anomalía es (problema lógico, computacional, de interfaz, de manejo de datos, de documentación, de calidad, entre otros). También puede referirse a qué tipo de mejoras se necesitan. Este atributo es obligatorio. En la figura 2 se presenta un árbol con la jerarquía de este atributo, mostrando en cada nodo la cantidad de valores posibles en ese nivel. No se presentan los nombres de cada valor por un tema de espacio físico.

Los atributos relacionados al Impacto que se deben clasificar dentro de la etapa de Investigación son los siguientes:

Prioridad Se refiere a qué prioridad tiene el defecto para ser solucionado. Este atributo es opcional.

Costo de Proyecto Se refiere a qué efecto causa en el presupuesto del proyecto el hecho de solucionar el defecto. Este atributo es obligatorio.

Calidad/Confiabilidad del Proyecto Se refiere a qué impacto causa en la calidad o confiabilidad del producto el hecho de solucionar el defecto. Este atributo es opcional.

Riesgo del Proyecto Se refiere al riesgo asociado a la implementación de la solución del defecto.

Cronograma del Proyecto Se refiere a qué efecto causa en el cronograma del proyecto el hecho de solucionar el defecto. Este atributo es obligatorio.

Sociedad Se refiere a qué impacto causa en la sociedad el hecho de implementar la solución. Este atributo es opcional.

3.4.3. Acción

Es cuando se establece un plan de acción, basado en los resultados de la etapa de Investigación, para solucionar el defecto y prevenir que el mismo ocurra en el futuro. El plan de acción debe incluir todas las actividades necesarias para resolver el defecto inmediatamente. También debe incluir actividades requeridas para revisar los procesos, políticas o cualquier otra condición necesaria para prevenir defectos similares.

Los atributos a clasificar dentro de la etapa de Acción son los siguientes:

Acción Correctiva Se refiere a qué hacer para prevenir que la anomalía ocurra de nuevo. Este atributo es opcional.

Resolución Se refiere a la acción a realizar para corregir el defecto. Este atributo es obligatorio.

3.4.4. Disposición

Es cuando se completan todas las acciones requeridas para solucionar el defecto o cuando se culmina de identificar todas las acciones correctivas a largo plazo.

El atributo a clasificar dentro de la etapa de Disposición es el siguiente:

Disposición Se refiere a qué sucedió para dar por cerrado el proceso para este defecto. Este atributo es obligatorio.

El valor posible para cada atributo es predeterminado, pero los mismos no se detallan aquí por cuestiones de espacio físico.

3.5. *Orthogonal Defect Classification*

Previo al año 1996, el análisis de defectos de software era generalmente efectuado mediante dos tipos de métodos: los *Statistical Defect Models* y los modelos *Root Cause Analysis*.

Los *Statistical Defect Models* son puramente cuantitativos (como los métodos matemáticos o estadísticos), abstractos, distantes del programador, de bajo costo y que pueden ser automatizados. Están restringidos a unos pocos dominios de aplicación.

Por otro lado, los modelos *Root Cause Analysis* son modelos de análisis puramente cualitativos (como los métodos utilizados en programas de prevención de defectos), concretos, se basan en la perspectiva del programador, costosos y necesitan muchos recursos humanos. Son utilizados en un amplio rango de dominios de aplicación.

ODC, *Orthogonal Defect Classification*, es una taxonomía de defectos desarrollada por IBM en 1996 [3]. La misma brinda un puente entre los dos extremos, *Statistical Defect Models* y *Root Cause Analysis*, mediante el desarrollo de un sistema de medición basado en semántica. Uno de los objetivos de ODC es brindar al equipo de desarrollo una retroalimentación constante durante el transcurso del proyecto. Además, ODC es utilizada para mejorar el proceso de desarrollo mediante la reducción de la cantidad de defectos con el avance del proyecto.

ODC propone dos pasos de recolección de información en el proceso de clasificación para diseño y código llamados Apertura y Clausura. El primer paso se ejecuta cuando un defecto es descubierto y un nuevo reporte de defecto es incorporado al sistema de registro y seguimiento de defectos. El otro paso se ejecuta cuando el defecto es detectado, corregido y el reporte de defecto se encuentra cerrado.

La taxonomía con sus atributos puede verse claramente en la figura 3. A continuacíón se describe los atributos de la taxonomía donde los mismos están organizados según los dos pasos del proceso.

3.5.1. Actividad

Se refiere a la actividad que se estaba realizando cuando se provocó la falla. En el caso de inspecciones, es la actividad que detecta el defecto directamente. Se mide en la fase de Apertura y los valores posibles son:

Inspección de Código Es una técnica formal de revisión de código cuyo objetivo principal es detectar e identificar anomalías en un producto de software. Consiste en el examen sistematizado por parte de pares del trabajo elaborado por una persona. Se buscan errores comunes, por lo cual se examina para detectar la presencia de errores más que simular la ejecución. Se utiliza una lista de errores a detectar, en la cual se incluyen los errores de programación clásica.

Inspección de Diseño Similar a la inspeccion de código, pero se realiza sobre los documentos de diseño.

Prueba Unitaria La prueba unitaria es una forma de verificar una unidad de software de forma aislada. Una unidad puede ser un método, una clase o un conjunto pequeño de clases que forman un módulo. Las pruebas unitarias son generalmente realizadas por los desarrolladores de la unidad.

Prueba de Integración Es aquella que se realiza una vez que se han aprobado las pruebas unitarias. El objetivo es verificar la comunicación entre dos o más componentes. Las pruebas de integración se focalizan en los

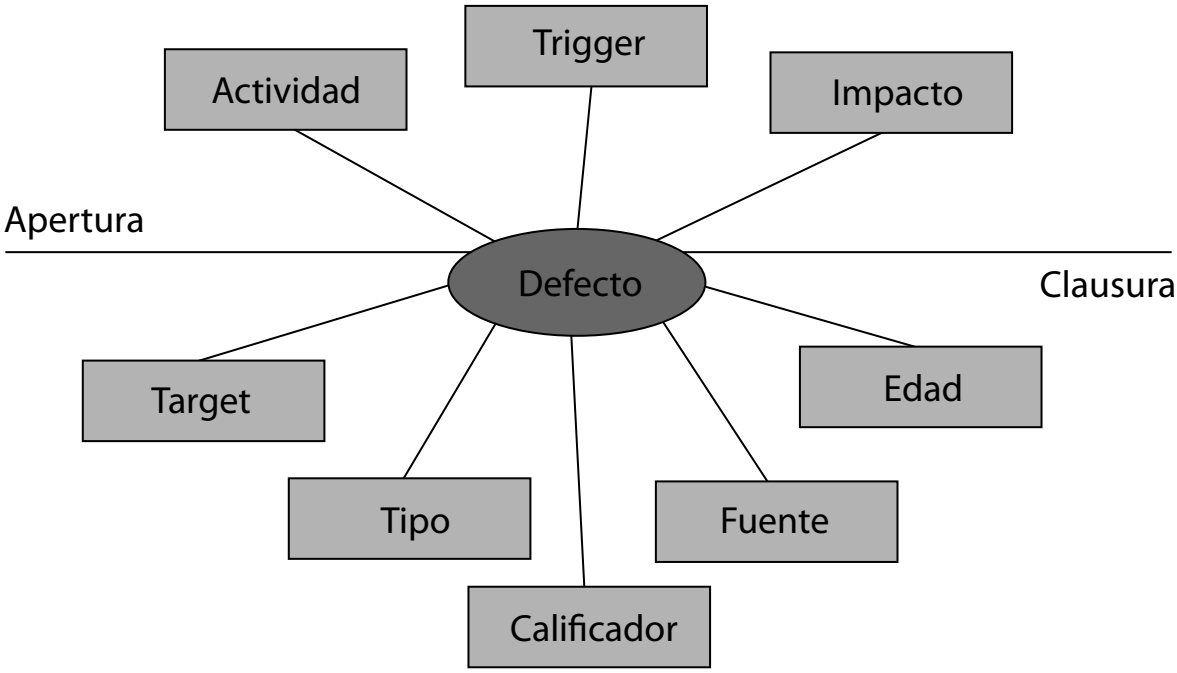

Figura 3. *Orthogonal Defect Classification*

defectos de las interfaces, y se utiliza en la integración de módulos, subsistemas o sistemas. Generalmente las pruebas de integración son realizadas por los desarrolladores de los componentes.

Prueba de Sistema Las pruebas de sistema tienen como objetivo verificar el sistema software en su totalidad, para comprobar si el mismo cumple con sus requerimientos. Abarca las pruebas funcionales, de usabilidad, de rendimiento, de seguridad, entre otras. Las pruebas pueden ser llevadas acabo por los desarrolladores del sistema o por un conjunto de verificadores expertos en el área con gran conocimiento de la especificación del software a testear.

3.5.2. *Trigger*

Es la actividad que provocó la ocurrencia de una falla. En el caso de inspecciones, se refiere a cómo se detectó el defecto. Este atributo es utilizado para proveer una medida de los aspectos de verificación del proceso de desarrollo y se mide en la fase de Apertura.

Hay tres clases de *triggers*, según las distintas actividades de verificación que son comúnmente empleadas en el desarrollo de software: Inspecciones, Pruebas Unitarias/Funcionales, Pruebas de Sistema. La distinción entre las mismas surge por cómo las mismas actúan para provocar fallas. Las Inspecciones son un proceso pasivo, ya que no hay nada que ejecutar. Las Pruebas Unitarias o Funcionales chequean activamente la implementación mediante la ejecución de código, impulsadas por *triggers* estructurales y de composición. Las Pruebas de Sistema emulan el uso en el entorno del cliente.

Los valores posibles para el atributo *Trigger* en Inspecciones son:

Compatibilidad con versiones Está relacionado con la comprensión de cómo la versión actual del producto funcionaría con versiones anteriores.

Compatibilidad Lateral Está relacionado con cómo el producto debería trabajar con otros productos con la misma configuración de software.

Conformidad del diseño Está relacionados con la completitud del diseño del producto respecto a los requerimientos y objetivos del producto.

Concurrencia Está relacionado con la comprensión de las cuestiones de *timing* y serialización relacionadas a la implementación del producto. Por ejemplo, mecanismos de bloqueo, regiones compartidas y secciones críticas.

Semántica Operacional Relacionado a la comprensión del flujo de la lógica respecto a la implementación de un diseño.

Consistencia/Completitud de Documento Está relacionado con la completitud de un diseño y asegurar la consistencia entre las distintas partes del diseño propuesto y la implementación.

Situaciones Raras Relacionado con implementaciones inusuales, idiosincrasias o información específica de un dominio que no es común.

Los valores posibles para el atributo *Trigger* en Pruebas Unitarias o Funcionales son:

Prueba de Cubrimiento Se refiere a la ejecución de una función a través de varias entradas para cubrir la mayor cantidad de casos posibles del dominio de parámetros. Es un *trigger* de prueba de caja negra.

Prueba Secuencial Estos son casos de pruebas que apuntan a ejecutar múltiples secciones del código con diferentes secuencias. También es de caja negra.

Prueba de Interacción Son pruebas que exploran interacciones más complicadas que generalmente no son cubiertas por secuencias simples.

Prueba de Variación Son pruebas que apuntan a ejecutar una única función usando múltiples entradas.

Cobertura de Camino Simple Son pruebas de caja blanca que apuntan a ejecutar los distintos caminos del código.

Cobertura de Combinación de Caminos Prueba de caja blanca que apunta a caminos de código más completos.

Los valores posibles para el atributo *Trigger* en Pruebas de Sistema son:

Recuperación/Manejo de Excepciones El defecto surge debido a un mal manejo de excepciones o no se ha ejecutado un proceso de recuperación.

Arranque y Reinicio del Sistema Esto se refiere a un producto siendo inicializado o reiniciado. Estos procedimientos pueden estar muy ligado a aplicaciones, como la base de datos.

Volumen de trabajo Esto indica que el producto alcanzó la capacidad máxima de alguno de los recursos.

Configuración de Hardware y Software Estos *triggers* son aquellos que son generados por cambios en el entorno, ya sea en hardware o software.

Modo Normal Esta categoría significa que para capturar estos *triggers*, no tuvo que suceder nada inusual. El producto falla cuando se suponía que debía funcionar normalmente.

3.5.3. Impacto

Se refiere al impacto que tiene en el producto la provocación de la falla. Se mide en la fase de Apertura y los valores posibles son:

Instalabilidad Problemas en la capacidad del producto software para ser instalado en un ambiente especificado.

Servicio El servicio brindado por el sistema no es el deseado por el cliente.

Standards El software no cumple completamente con algún standard planteado en los requerimientos.

Integridad/Seguridad Problemas en la capacidad del producto software para proteger la información y los datos de modo que las personas o los sistemas no autorizados no puedan leerlos o modificarlos, y a las personas o los sistemas autorizados no se les niegue el acceso a ellos.

Migración Se refiere a problemas al exportar/importar datos desde/hacia el sistema.

Confiabilidad Problemas en la capacidad del producto software para mantener su nivel de desempeño cuando es utilizado bajo condiciones específicas (durante un determinado período de tiempo). Esto incluye problemas en la madurez, tolerancia a fallas y recuperabilidad del sistema.

Performance Se refiere a problemas de sobrecarga o estrés.

Documentación Se refiere a cualquier error en la documentación, ya sea que la misma no se encuentra acorde a la implementación o que la misma no esté completa.

Requerimientos No se cumplen todos los requerimientos especificados por el cliente.

Mantenimiento Problemas en la capacidad del producto software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptaciones del software a cambios de ambiente y en requisitos o especificaciones funcionales.

Usabilidad Se refiere a problemas en la capacidad de un software para ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.

Accesibilidad Se refiere a problemas para acceder a distintas funcionalidades o datos por parte de personas con algún tipo de discapacidad.

Capacidad Se refiere a problemas en la capacidad del sistema para realizar alguna funcionalidad.

3.5.4. *Target*

Se refiere a qué entidad de alto nivel debió ser corregida, es decir, la ubicación del defecto. Se mide en la fase de Clausura y los valores posibles son:

Requerimientos Se corrigieron los documentos de requerimientos y especificaciones de software.

Diseño Se corrigieron los documentos de diseño y arquitectura.

Código La implementación de código fue lo que se tuvo que corregir.

Build/Package/Merge Se tuvieron que corregir problemas en el proceso de *build*, en librerías de sistema o en el manejo de control de versiones.

Información Se debió corregir los manuales de usuario, manuales de instalación, ayuda en linea o mensajes al usuario.

Interfaz de Usuario Se debieron hacer cambios estéticos en la interfaz de usuario.

3.5.5. Fuente

Se refiere a quién desarrolló el *target*. Se mide en la fase de Clausura y los valores posibles son:

En casa El defecto se encuentra en un área en la que trabajó un equipo de desarrollo de la organización.

Librería El defecto se encuentra usando una librería. El problema puede haber sido por usar mal la librería o un defecto de la librería en sí.

Externo El defecto se encuentra en una parte que proviene de un tercero.

Ported El defecto está relacionado con el uso de algo que fue validado para un entorno distinto al utilizado.

3.5.6. Edad

Se refiere a cuál es el historial del *target*. Se mide en la fase de Clausura y los valores posibles son:

Base El defecto se encuentra en una parte del producto que no ha sido modificada para el proyecto actual y no es parte de una librería standard. El defecto no fue introducido por el proyecto actual, sino que es un defecto latente.

Nuevo El defecto se encuentra en una función que ha sido creada e introducida en el proyecto actual.

Re-escrito El defecto fue introducido como resultado de rediseño y/o reimplementación de una función vieja, con motivo de mejorar su diseño o calidad.

Re-corregido El defecto fue introducido al corregir un defecto anterior.

3.5.7. Tipo de Defecto

Se refiere a qué debió ser corregido. Se mide en la fase de Clausura y los valores posibles son:

Asignación Un defecto de este tipo implica el cambio de unas pocas líneas de código, como la inicialización de un bloque de control o de estructuras de datos.

Chequeo Se refiere a lógica que ha fallado en la validación de datos, chequeo de valores antes de ser utilizados o condición de iteraciones.

Algoritmo Son defectos de eficiencia o correctitud que pueden ser corregidos mediante la re-implementación del algoritmo o mediante un cambio de la estructura de los datos locales, sin necesidad de un cambio en el diseño.

Función Un defecto de este tipo es aquel que afecta significativamente la capacidad, las características requeridas para el usuario final, la API, las interfaces con hardware o estructuras globales. Requiere un cambio formal en el diseño.

Timing/Serialization Son aquellos defectos que se corrigen mediante la mejora en el manejo de recursos compartidos y de tiempo real.

Interfaz Corresponde a defectos en la interacción con otros componentes, módulos, controladores de dispositivos, llamados a funciones, bloques de control o lista de parámetros.

Build/Package/Merge Se refiere a defectos que ocurren debido a defectos en bibliotecas, control de cambios o control de versión.

Documentación Son defectos que pueden afectar a las publicaciones o las notas de mantenimiento.

3.5.8. Calificador de Defecto

Explica cómo se inyectó el defecto. Se mide en la fase de Clausura y los valores posibles son:

Faltante El defecto es causado por algo que fue omiso. Por ejemplo, falta una sentencia de asignación.

Incorrecto El defecto es causado por algo hecho incorrectamente. Por ejemplo, el chequeo de datos realizado usa valores incorrectos.

Extraño El defecto se debe a algo que no es relevante o pertinente al documento o al código en sí. Por ejemplo, en el documento de diseño hay una sección que no es necesaria para el producto actual, por lo que debería ser quitada.

3.6. Taxonomía de Boris Beizer

En el libro *Software Testing Techniques, Second Edition* [1], Boris Beizer presenta una taxonomía jerárquica de tipos de defectos de software. Los defectos se clasifican por un número de 4 dígitos y a veces incluye subnumeración usando el punto ".". Además la letra "x" se utiliza de forma de posibilitar la expansión de la taxonomía. El último dígito de cada conjunto es el 9, que se utiliza cuando el defecto no coincide con ninguna de las categorías existentes o porque no existe una descomposición más fina.

En cada subsección se presenta una categoría básica y las subcategorías correspondientes. Se puede profundizar en la taxonomía y ver ejemplos de cada categoría en [11].

3.6.1. 1xxx - Requerimientos y Características

Los defectos de este tipo son aquellos que se refieren a la especificación de requerimientos y características. En la figura 4 puede verse un resumen de la categoría 1xxx. A continuación se detallan sus subcategorías.

11xx - Requerimiento incorrecto El requerimiento o una parte del mismo no es correcto.

- **111x - Incorrecto** El requerimiento está mal.
- **112x - No deseado** El requerimiento está correctamente definido, pero no es deseable.
- **113x - Innecesario** El requerimiento no es necesario.

12xx - Lógica El requerimiento es ilógico o no es razonable.

- **121x - Ilógico** Defecto ilógico, usualmente debido a una contradicción interna que puede ser expuesta por un análisis lógico de casos.

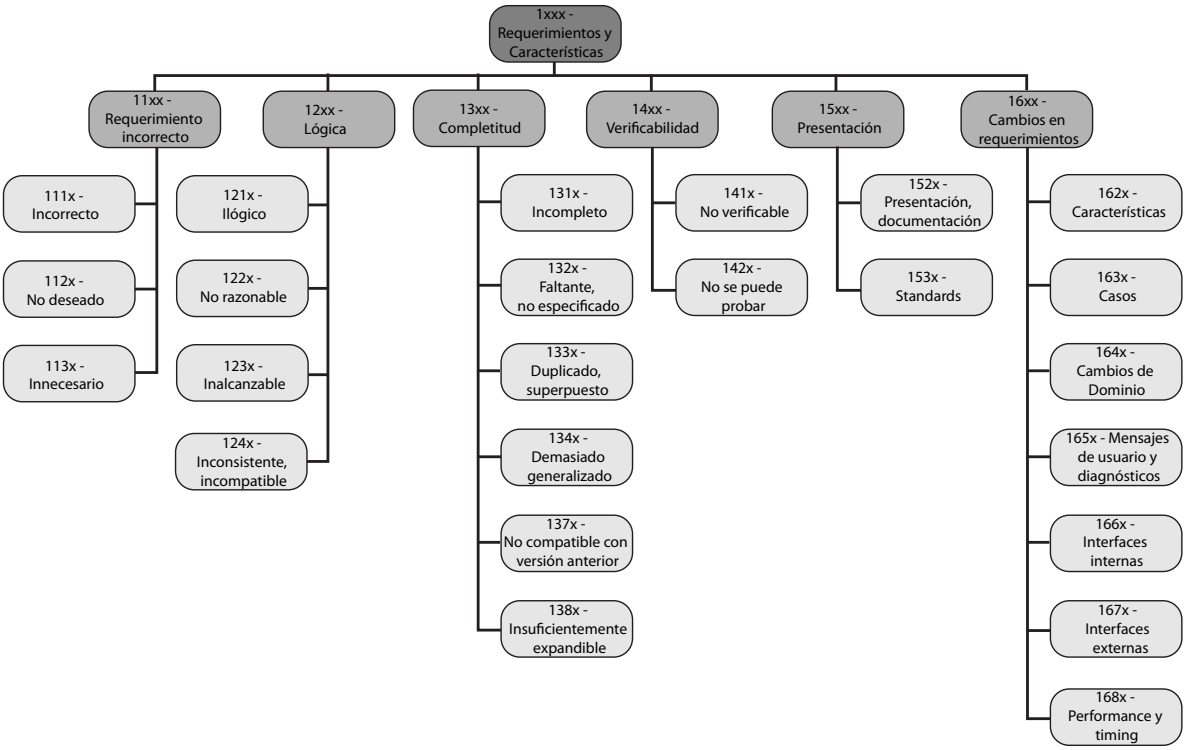

Figura 4. 1xxx - Requerimientos y Características - Taxonomía de Beizer

122x - No razonable Lógicamente correcto y consistente, pero no razonable con respecto al ambiente, presupuesto o limitaciones de tiempo del proyecto.

123x - Inalcanzable Requerimiento fundamentalmente imposible o que no puede ser alcanzado bajo las limitaciones existentes.

124x - Inconsistente, incompatible El requerimiento es inconsistente con otros requerimientos o con el entorno.

13xx - Completitud El requerimiento está especificado de manera ambigua, incompleta o demasiado especificado.

131x - Incompleto La especificación es incompleta. Casos, características, variaciones o atributos sin especificar y por lo tanto no implementados.

132x - Faltante, no especificado El requerimiento entero está sin especificar.

133x - Duplicado, superpuesto El requerimiento se superpone parcial o totalmente con otro requerimiento ya implementado o especificado.

134x - Demasiado generalizado El requerimiento está especificado de forma correcta y consistente pero demasiado generalizado para la aplicación.

137x - No compatible con versión anterior El requerimiento especificado implica que objetos creados o manipulados por versiones anteriores no puedan ser procesados correctamente por esta versión.

138x - Insuficientemente expandible El requerimiento especificado no puede ser expandido en la manera que es requerido.

14xx - Verificabilidad Defectos en la especificación relacionados con la verificación de que los requerimientos están correctamente implementados.

141x - No verificable El requerimiento, si se implementa, no puede ser verificado por motivos de tiempo o presupuesto.

142x - No se puede probar No es posible diseñar y/o ejecutar pruebas para verificar el requerimiento.

15xx - Presentación Defectos en la presentación o documentación de requerimientos. Los requerimientos se asumen correctos, pero la forma en que están presentados no.

152x - Presentación, documentación Defecto en la presentación general, documentación o formato.

153x - Standards La presentación viola los standards para requerimientos.

16xx - Cambios en requerimientos Los requerimientos han cambiado entre el momento en que se comenzó a implementar y el momento en que terminó la verificación.

162x - Características Los cambios en los requerimientos se deben a cambios en las características.

163x - Casos Los cambios en los requerimientos se deben a cambios en los casos de uso.

164x - Cambios de Dominio El dominio de los datos de entrada ha sido moficado.

165x - Mensajes de usuario y diagnósticos Cambios en texto, contenido o condiciones en los mensajes al usuario como *prompts*, advertencias o mensajes de error.

166x - Interfaces internas Interfaces internas al sistema han sido cambiadas.

167x - Interfaces externas Interfaces externas, como *drivers* de dispositivos, han sido cambiadas.

168x - Performance y *timing* Cambios en los requerimientos de performance y/o *timing*.

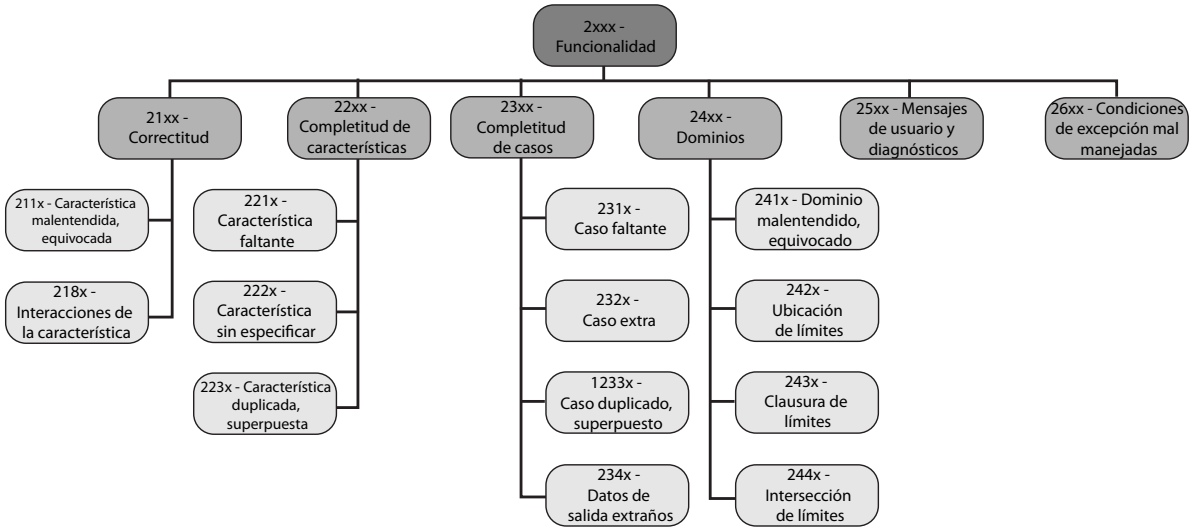

Figura 5. 2xxx - Funcionalidad - Taxonomía de Beizer

3.6.2. 2xxx - Funcionalidad

Se refiere a defectos en la implementación de características o funcionalidades del software. En la figura 5 puede verse un resumen de la categoría 2xxx. A continuación se detallan sus subcategorías.

21xx - Correctitud Defectos en la correctitud de la implementación.

- **211x - Característica malentendida, equivocada** La implementación de la característica no es correcta respecto a la especificación. El defecto se debe a que la característica no se entendió, o se entendió pero se implementó mal.
- **218x - Interacciones de la característica** La característica está implementada correctamente en sí misma, pero tiene interacciones incorrectas con otras características, o la interacción especificada o implicada se maneja de manera incorrecta.

22xx - Completitud de características Defectos relacionados con la completitud de las características que están implementados.

- **221x - Característica faltante** No se implementa una característica entera.
- **222x - Característica sin especificar** Una característica no especificada ha sido implementada.
- **223x - Característica duplicada, superpuesta** La implementación de la característica duplica o se superpone a características implementadas por otras partes del software.

23xx - Completitud de casos Defectos relacionados con la completitud de los casos de las características.

- **231x - Caso faltante** No se implementa un caso entero.
- **232x - Caso extra** Son casos que no debieron ser manejados.
- **233x - Caso duplicado, superpuesto** Manejo duplicado de casos o superposición parcial con otros casos.
- **234x - Datos de salida extraños** Hay salida de datos que no son requeridos.

24xx - Dominios El procesamiento del caso o la característica depende de una combinación de valores de entrada. Un defecto de dominio existe si un procesamiento incorrecto es ejecutado para una combinación de valores de entrada. El procesamiento se asume correcto.

- **241x - Dominio malentendido, equivocado** No entender el tamaño, forma, límites, o otras características del domino de entrada especificado para el caso o la característica. La mayoría de los defectos relacionados con el manejo de casos extremos son defectos de dominio.
- **242x - Ubicación de límites** Los valores o expresiones que definen un límite en el domino son erróneos.
- **243x - Clausura de límites** Puntos finales y límites del dominio son asociados incorrectamente con un domino adyacente.
- **244x - Intersección de límites** Los límites del dominio son definidos por una relación entre variables de control del dominio. La implementación de la relación es incorrecta.

25xx - Mensajes de usuario y diagnósticos *Prompts*, listados u otra forma de comunicación con el usuario es incorrecta. El procesamiento se asume correcto. Defectos que caen dentro de esta subcategoría son: Advertencia falsa, fracasar al advertir, mensaje incorrecto, ortografía, formatos.

26xx - Condiciones de excepción mal manejadas Condiciones de excepción como problemas de recursos o modos de falla, que requieren un manejo especial, no son manejadas correctamente o se utilizan mecanismos incorrectos de manejo de excepciones.

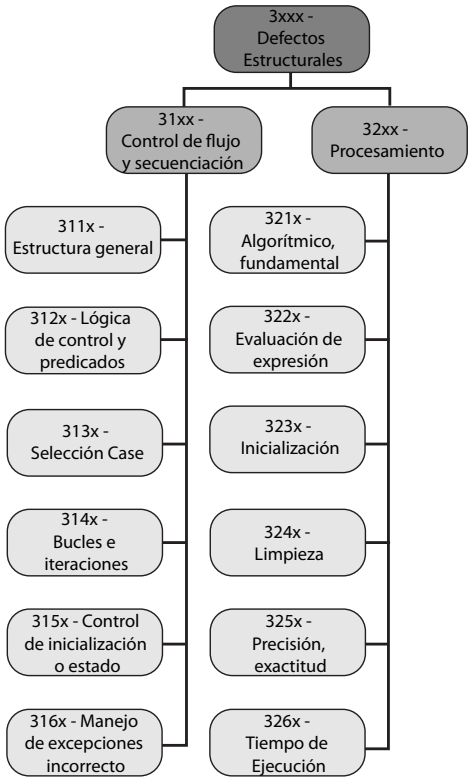

Figura 6. 3xxx - Defectos Estructurales - Taxonomía de Beizer

3.6.3. 3xxx - Defectos Estructurales

Se refiere a defectos relacionados con la estructura de los componentes. En la figura 6 puede verse un resumen de la categoría 3xxx. A continuación se detallan sus subcategorías.

31xx - Control de flujo y secuenciación Defectos específicamente relacionados con el flujo de control del programa.

- **311x - Estructura general** Defectos generales relacionados con la estructura del componente. Entran en esta subcategoría los defectos por camino inaccesible, código inalcanzable y código muerto.
- **312x - Lógica de control y predicados** El camino ejecutado a través de un programa es dirigido por predicados de flujo de control. Esta categoría trata defectos en la implementación de estos predicados.
- **313x - Selección *Case*** Defectos simples en la selección de casos, como ser expresión de selección de casos formulada de forma incorrecta, lista de *switch-case*, defecto en la asignación de un *switch-case*.
- **314x - Bucles e iteraciones** Defectos relacionados con el control de loops.
- **315x - Control de inicialización o estado** Defectos relacionados con la inicialización del flujo de control del programa y con cambios de estado que afectan el flujo de control.
- **316x - Manejo de excepciones incorrecto** Toda invocación incorrecta de un manejador de excepciones de flujo de control, no previamente categorizada.

32xx - Procesamiento Defectos relacionados con el procesamiento bajo la suposición de que el flujo de control es correcto.

- **321x - Algorítmico, fundamental** Algoritmo seleccionado es inadecuado, pero utilizado correctamente.
- **322x - Evaluación de expresión** Defectos relacionados con la manera en la que se evalúan las expresiones artiméticas, booleanas, de strings.
- **323x - Inicialización** Defectos en la inicialización de variables, expresiones, funciones usadas en procesamiento (auxiliares). No se incluyen los defectos de inicialización asociados con declaraciones y sentencias de datos (413x) ni inicialización de loops (314x).
- **324x - Limpieza** Manejo incorrecto de la limpieza de áreas de datos, registros y estados asociados con el procesamiento.
- **325x - Precisión, exactitud** Precisión insuficiente o excesiva, exactitud insuficiente y otros defectos relacionados al sistema de representación de números usado.
- **326x - Tiempo de Ejecución** Tiempo de ejecución excesivo para el procesamiento de un componente.

3.6.4. 4xxx - Datos

Se refiere a defectos en la definición, estructura o uso de datos. En la figura 7 puede verse un resumen de la categoría 4xxx. A continuación se detallan sus subcategorías.

41xx - Definición de datos, estructura, declaración Defectos en la definición, estructura e inicialización de datos. Esta categoría se aplica si el objeto es declarado estáticamente en el código original o creado dinámicamente.

- **411x - Tipo** El tipo de objeto de datos, en su declaración, es incorrecto.
- **412x - Dimensión** Para arreglos y otros objetos que tienen una dimensión, como por ejemplo arreglos, registros y archivos. Un defecto en la dimensión o en las dimensiones mínimas o máximas, o en declaraciones de redimensión.
- **413x - Valores iniciales o por defecto** Defectos en los valores iniciales asignados al objeto de datos, selección de valores por defecto incorrectos, o fallar al suministrar un valor por defecto en caso de ser necesario.

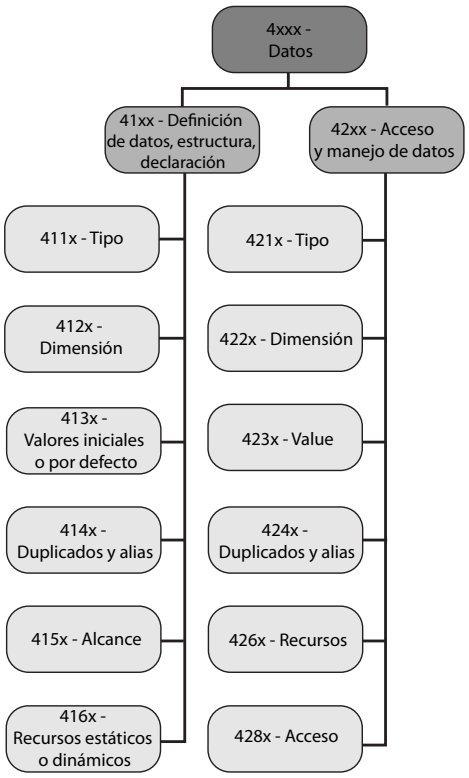

Figura 7. 4xxx - Datos - Taxonomía de Beizer

414x - Duplicados y alias Defectos relacionados a la duplicación incorrecta o la falla al crear un objeto duplicado.

415x - Alcance El alcance, partición o componente al cual aplica el objeto está especificado de forma incorrecta.

416x - Recursos estáticos o dinámicos Relacionado con la declaración de recursos asignados estática y dinámicamente.

42xx - Acceso y manejo de datos Relacionado con el acceso y la manipulación de objetos de datos que se asumen correctamente definidos.

421x - Tipo Defectos relacionados con el tipo del objeto.

422x - Dimensión Para dimensiones dinámicamente variables de un objeto con dimensión. Se refiere a un defecto en la dimensión. Por ejemplo redimensión dinámica de arreglos, exceder longitud máxima de un archivo, remover uno más que el número mínimo de registros.

423x - Value Relacionado con el valor de los objetos de datos.

424x - Duplicados y alias Defectos en duplicación y alias dinámico (en tiempo de ejecución) de objetos.

426x - Recursos Relacionado con recursos asignados dinámicamente y *pools* de recursos, en cualquier medio de memoria que exista: principal, cache, disco, RAM.

428x - Acceso Relacionado con el acceso a los objetos, no la manipulación de los mismos. En este contexto, accesos incluyen: leer, escribir, modificar, (y en algunos casos) crear y destruir.

3.6.5. 5xxx - Implementación y Codificación

Se refiere a defectos en la implementación de software y aplicación de standards. Los defectos que caen en esta categoría son por ejemplo, datos sin declarar, rutinas sin declarar, código, problemas de inicialización. La mayoría de ellos son capturados por un buen compilador. También se incluyen defectos de documentación como comentarios erróneos, ya que los mismos conducen a acciones incorrectas en el mantenimiento, causando la inclusión de otros defectos. En la figura 8 puede verse un resumen de la categoría 5xxx. A continuación se detallan sus subcategorías.

51xx - Codificación y Tipografía Defectos que pueden ser claramente atribuidos a problemas de codificación o tipografía. La clasificación para un defecto en esta categoría es subjetiva.

511x - Codificación apurada, tipográfico Defectos que son razonablemente atribuidos a errores de tipeo u otros defectos tipográficos.

512x - Instrucción malentendida Defectos que son razonablemente atribuidos a un malentendido de la operación de una instrucción.

52xx - Violación de standards Defectos que se refieren a violación o malentendidos en la aplicación de standards y convenciones. Se asume que el software funciona correctamente.

521x - Violación de estructuras Violaciones en las estructuras del flujo de control u organización del software.

522x - Definición de datos, declaraciones La ubicación de la declaración de objetos de datos no está acorde a los standards.

523x - Acceso a datos Violación en las convenciones de cómo los objetos de datos de distinto tipo son accedidos.

524x - Llamado e invocación Defectos en la forma en que otros procesos son llamados, invocados o en la forma de comunicación con los mismos.

526x - Mnemotécnicos, etiquetas Violación a las reglas de asignación de nombres a objetos.

528x - Comentarios Violación de las convenciones que gobiernan el uso, ubicación, densidad y formato de los comentarios.

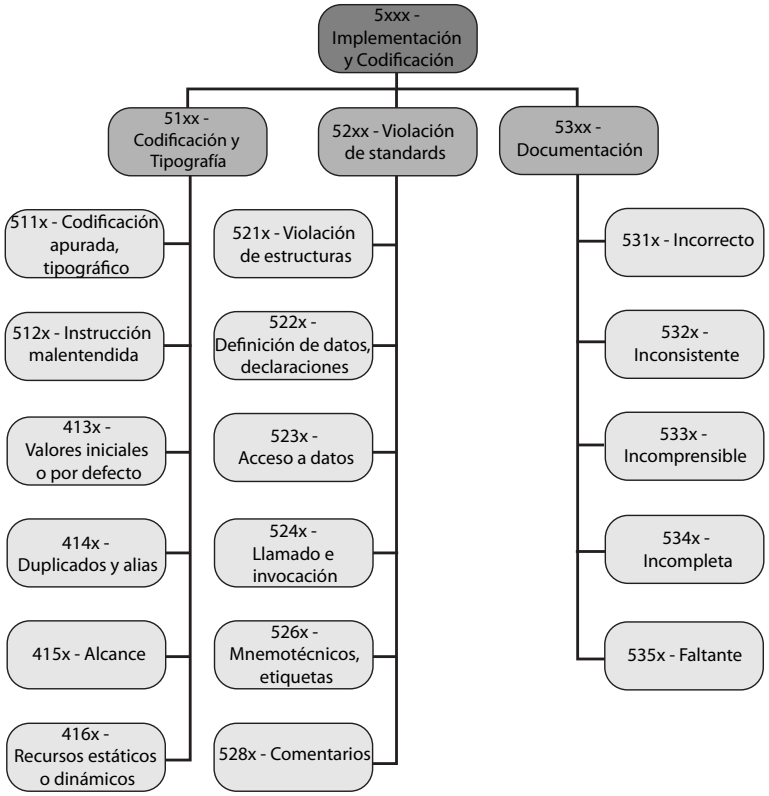

Figura 8. 5xxx - Implementación y Codificación - Taxonomía de Beizer

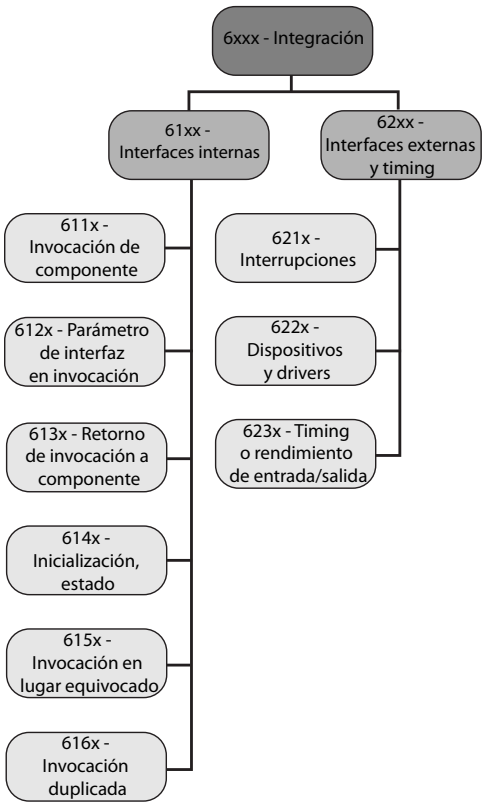

Figura 9. 6xxx - Integración - Taxonomía de Beizer

53xx - Documentación Defectos en la documentación asociados al código o al contenido de los comentarios incluidos en el código.

531x - Incorrecto Frase incorrecta en la documentación.

532x - Inconsistente Frase en la documentación es inconsistente consigo misma o con otras frases.

533x - Incomprensible La documentación no puede ser comprendida por un lector calificado.

534x - Incompleta La documentación es correcta, pero información importante no está contemplada.

535x - Faltante Falta gran parte de la documentación.

3.6.6. 6xxx - Integración

Se refiere a defectos en la integración de interfaces con componentes. Los componentes se asumen correctos. En la figura 9 puede verse un resumen de la categoría 6xxx. A continuación se detallan sus subcategorías.

61xx - Interfaces internas Defectos relacionados con las interfaces entre componentes que se comunican con el programa. Se asume que los componentes son correctos. En este contexto, la transferencia directa o indirecta de datos o información de control mediante objetos de memoria como tablas, recursos asignados dinámicamente, o archivos, constituyen una interfaz interna.

611x - Invocación de componente Defectos relacionados a la invocación de componentes de software. En este sentido un componente puede ser una subrutina, función, macro, programa, segmento de programa, o cualquier otro componente de procesamiento.

612x - Parámetro de interfaz en invocación Relacionado con los parámetros de la invocación, su número, orden, tipo, posición y valores.

613x - Retorno de invocación a componente Relacionado con la interpretación de los parámetros proporcionados por el componente invocado, en retorno al componente invocador, o sobre el pase del control a algún otro componente. En este contexto, un registro, una secuencia de retorno de una subrutina, o un archivo pueden calificar dentro de esta categoría de defectos. Los defectos incluidos aquí no son defectos en el componente que creó los datos de retorno, sino en la manipulación posterior y la interpretación de los datos en el componente que los recibe.

614x - Inicialización, estado Componente invocado no inicializado, o inicializado en un estado erróneo, o con datos incorrectos.

615x - Invocación en lugar equivocado El lugar o estado en el que se invoca a un componente en el componente invocador es incorrecto.

616x - Invocación duplicada El componente no debería haber sido invocado o ha sido invocado más a menudo que lo necesario.

62xx - Interfaces externas y *timing* Defectos relacionados con las interfaces externas, como dispositivos de entrada/salida y/o *drivers* u otro software que no opera bajo la misma estructura de control.

621x - Interrupciones Manejo o seteo incorrecto de interrupciones.

622x - Dispositivos y *drivers* Interfaz incorrecta con dispositivos o *drivers*, o interpretación incorrecta de la información de estado retornada.

623x - *Timing* o rendimiento de entrada/salida Defectos en el timing con dispositivos externos.

3.6.7. 7xxx - Sistema y Arquitectura de Software

Se refiere a defectos que no se pueden atribuir a un componente o interfaces entre componentes, pero afectan el sistema, la arquitectura de software o la arquitectura de hardware. En la figura 10 puede verse un resumen de la categoría 7xxx. A continuación se detallan sus subcategorías.

71xx - Sistema Operativo Defectos relacionados al uso de las funcionalidades del sistema operativo. No son defectos en el sistema operativo en sí.

711x - Invocación, comando Se envió un comando erróneo al sistema operativo o se invocó una funcionalidad incorrectamente.

712x - Datos retornados, malinterpretación de estado Se ignoran o malinterpretan los datos devueltos por el sistema operativo o la información de estado.

714x - Espacio Recurso de memoria requerido no está disponible o se pidió de manera incorrecta.

72xx - Arquitectura de software Problemas en la arquitectura no definidos anteriormente.

721x - Bloqueos y semáforos Defectos en el uso de mecanismos de bloqueo y funcionalidades de comunicación interprocesos. No son defectos en los mecanismos en sí.

722x - Prioridad Defectos relacionados a la prioridad de la tarea.

723x - Transacción, control de flujo Defectos relacionados a la definición de flujos de control.

724x - Manejo de recursos y control Defectos relacionados al manejo de recursos compartidos.

725x - Llamados recursivos Defectos en el uso de invocaciones recursivas de componentes de software.

726x - Reentrada Defectos relacionados a la reentrada de componentes del sistema.

73xx - Recuperación Defectos relacionados a la recuperación de objetos luego de una falla.

74xx - Performance Defectos relacionados al rendimiento, retrasos en el comportamiento del software, asumiendo que todos los otros aspectos están correctos.

741x - Rendimiento inadecuado El sistema no tiene el rendimiento esperado.

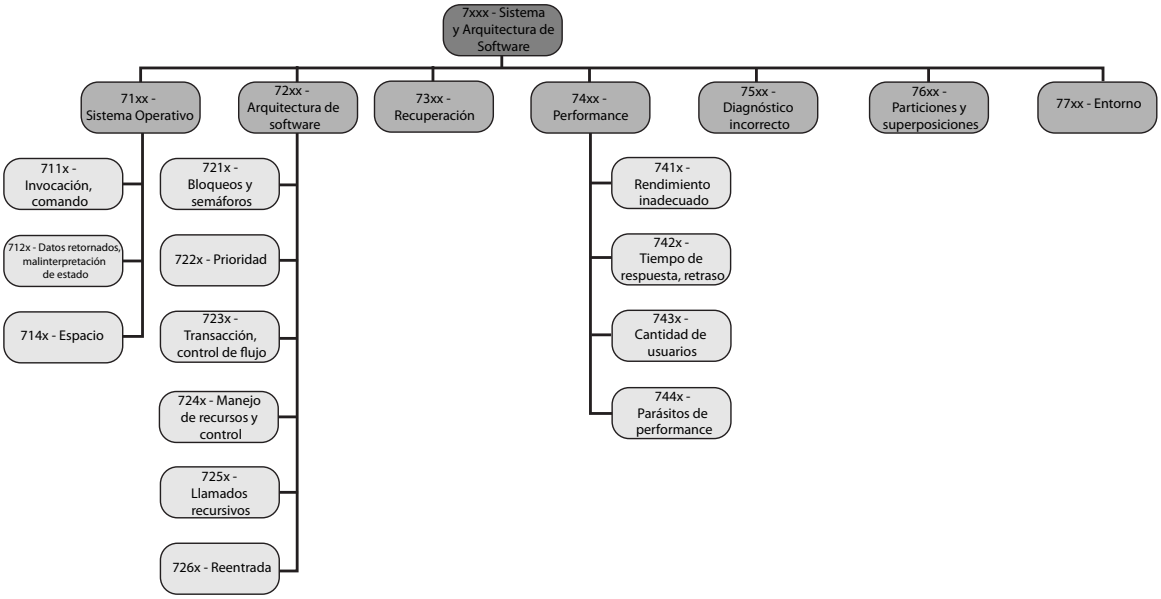

Figura 10. 7xxx - Sistema y Arquitectura de Software - Taxonomía de Beizer

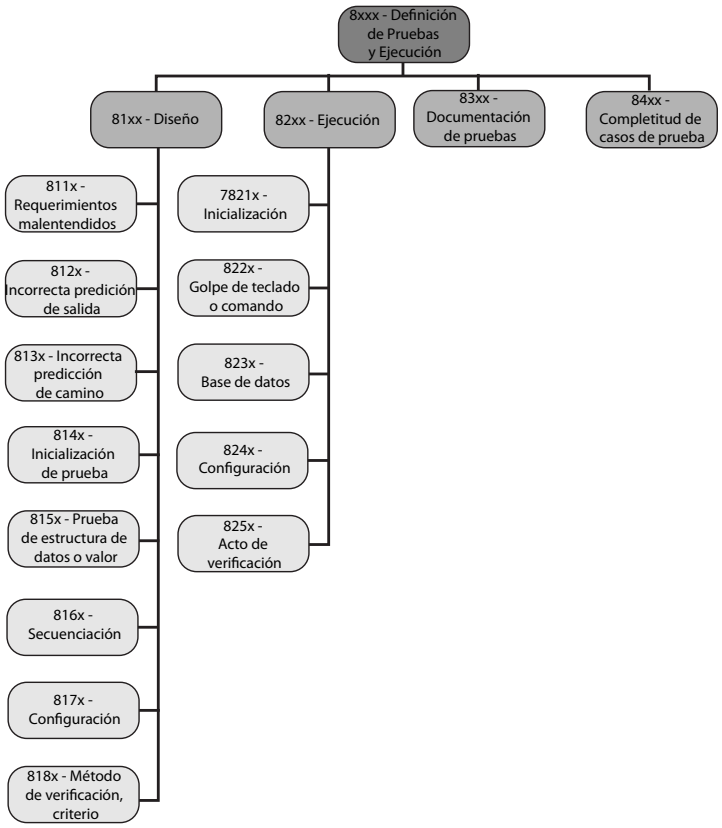

Figura 11. 8xxx - Definición de Pruebas y Ejecución - Taxonomía de Beizer

742x - Tiempo de respuesta, retraso El tiempo de respuesta de los eventos no es el especificado, retraso en los eventos.

743x - Cantidad de usuarios El número máximo especificado de usuarios simultáneos o tareas simultáneas, no son coherentes con los retrasos especificados en las transacciones.

744x - Parásitos de performance Defecto cuyo síntoma principal es la degradación de la performance.

75xx - Diagnóstico incorrecto Diagnóstico o mensaje de error incorrecto. La invocación al manejador de excepciones fue incorrecta.

76xx - Particiones y superposiciones La memoria o memoria virtual está incorrectamente particionada, superpone un área incorrecta o hay conflicto de particiones.

77xx - Entorno Versión equivocada de sistema operativo, generación del sistema incorrecta u otro defecto en el entorno del *host*.

3.6.8. 8xxx - Definición de Pruebas y Ejecución

Se refiere a defectos en la definición, diseño, ejecución de pruebas y manejo de datos usados durante las pruebas. En la figura 11 puede verse un resumen de la categoría 8xxx. A continuación se detallan sus subcategorías.

81xx - Diseño Defectos en el diseño de las pruebas.

811x - Requerimientos malentendidos Las pruebas no se corresponden con los componentes porque el verificador que diseñó la prueba no comprendió los requerimientos.

812x - Incorrecta predición de salida La salida predecida de la prueba no se corresponde con la salida requerida.

813x - Incorrecta predicción de camino La salida predecida de la prueba es correcta, pero se llega a la misma por un camino predecido incorrectamente.

814x - Inicialización de prueba Las condiciones iniciales especificadas para la prueba son incorrectas.

815x - Prueba de estructura de datos o valor Los objetos de datos utilizados o sus valores están incorrectos.

816x - Secuenciación La secuencia en que las pruebas son ejecutadas, en relación a otras pruebas o la inicialización de la prueba, es incorrecta.

817x - Configuración El hardware, la configuración de software o entorno especificado para la prueba es incorrecto.

818x - Método de verificación, criterio El método por el cual la salida va a ser verificada es incorrecto o imposible.

82xx - Ejecución Defectos en la ejecución de pruebas.

821x - Inicialización El componente probado no fue inicializado al estado correcto o con los valores correctos.

822x - Golpe de teclado o comando Un simple golpe de teclado o botón lanza error.

823x - Base de datos La base de datos usada en la prueba era incorrecta.

824x - Configuración La configuración o el entorno especificado para la prueba no fue usado durante la ejecución.

825x - Acto de verificación El acto que verifica la salida fue ejecutado incorrectamente.

83xx - Documentación de pruebas La documentación de los casos de prueba o criterios de verificación es incorrecta o confusa.

84xx - Completitud de casos de prueba Faltan casos de prueba para cubrir el criterio especificado.

3.6.9. 9xxx - Otros

Se refiere a todo defecto que no clasifica en las categorías anteriores.

3.7. Mapeo de Taxonomías por Freimut

En el Cuadro 1 se presenta el mapeo realizado por Freimut entre los atributos presentados en la meta taxonomía y los atributos existentes en las taxonomías ODC, HP e IEEE.

Analizando los atributos de ODC, observamos que el atributo Edad se encuentra bajo el meta atributo Ubicación. Nosotros no coincidimos con esto ya que, si bien la Edad se refiere a la historia del Target, un cambio en el valor del atributo Edad no implica un cambio en la ubicación del defecto. Respecto al atributo Tipo, el mismo se encuentra bajo el meta atributo Mecanismo, lo cual no nos resulta adecuado ya que el Tipo no describe cómo el defecto fue corregido. Tampoco coincidimos con la categorización del atributo Actividad bajo el meta atributo Mecanismo, ya que la Actividad es un atributo que se mide en la etapa de Apertura y se refiere a la actividad que provocó la falla, y eso es distinto a la actividad que detectó el defecto.

Analizando los atributos de HP, observamos que el atributo Tipo se encuentra bajo el meta atributo Síntoma. Nosotros no estamos de acuerdo en esto ya que el Tipo no describe lo que se observa cuando se provoca la falla y tampoco describe la actividad que provoca la falla.

Analizando los atributos la taxonomía presentada por la IEEE, observamos lo mismo que planteamos para el atributo Tipo de HP. Tampoco coincidimos con la categorización del atributo Actividad del Proyecto bajo el meta atributo Mecanismo, ya que la Actividad del Proyecto es un atributo que se mide en la etapa de Reconocimiento y se refiere a lo que se estaba realizando cuando se provocó la falla, y eso es distinto a la actividad que detectó

el defecto. Respecto a los atributos Causa sospechada y Causa actual, observamos que los mismos se encuentran bajo el meta atributo Causa (Error). Esta categorización no nos resulta adecuada, ya que el meta atributo Causa (Error) se refiere a errores humanos que llevaron a la introducción del defecto, y los atributos de la IEEE tienen un enfoque de ubicación. Finalmente se observa que existen atributos que no están reflejados en el mapeo y deberían estarlo, tal es el caso del atributo Gravedad que debería estar bajo el meta atributo Gravedad y el atributo Cronograma del Proyecto que debería estar bajo el meta atributo Costo.

Atributos	**ODC**	**HP**	**IEEE**
Ubicación	Target, Edad	Origen	Fuente
Tiempo			Fase del Proyecto
Síntoma	Trigger	Tipo	Tipo, Síntoma, Estado del Producto
Resultado final	Impacto		Estado del Producto, Repetibilidad
Mecanismo	Tipo, Actividad, Calificador	Modo	Actividad del Proyecto, Acción correctiva
Causa (Error)			Causa sospechada, Causa actual
Gravedad			
Costo			

Cuadro 1. Mapeo entre atributos de la meta taxonomía y de taxonomías, presentado por Freimut

4. Comparación Teórica

Las secciones anteriores se enfocaron en la presentación de las estructuras, componentes y propiedades de las taxonomías de defectos de software, y en la descripción de varias taxonomías utilizadas tanto en la industria como en el ámbito académico. El objetivo esta sección es presentar nuestro marco de comparación teórico. Además, se realizará una evaluación de las taxonomías ya vistas contra el marco presentado.

En la literatura solamente se encontró una comparación realizada por Freimut [4], la cual no profundiza demasiado. Sin embargo, usamos dicha comparación para crear un marco de comparación teórico original que se presenta en la sección 4.1. La comparación propiamente dicha de todas las taxonomías presentadas en el capítulo anterior, se realizará en la sección 4.2.

4.1. Marco Teórico

Para poder comparar teóricamente las taxonomías de una forma correcta y coherente, primero es necesario definir un marco para cuantificar objetiva y homogéneamente las características, fortalezas y deficiencias de cada taxonomía. El tener definido un marco no sólo permite realizar un análisis comparativo de las taxonomías vistas anteriormente, sino que sirve para comparar cualquier taxonomía de defectos de software.

La comparación que realiza Freimut es interesante ya que la misma plantea observar los atributos presentes en distintas taxonomías incluyendo HP, IEEE y ODC. De todas formas no resulta completa para alcanzar nuestra meta, ya que no sólo se puede caracterizar una taxonomía por sus atributos, sino también por sus propiedades. Además, su comparación no incluye todas las taxonomías presentadas ni es totalmente coherente con sus definiciones, como puede observarse en la sección 3.7.

Para poder cumplir con el objetivo de realizar un balance teórico completo y correcto, proponemos el desarrollo de un marco de comparación. El mismo está compuesto por dos vistas: Atributos y Propiedades.

La vista basada en Atributos toma como idea inicial la propuesta planteada por Freimut, la cual modificamos mediante la incorporación de atributos más específicos. El objetivo de esta vista es evaluar la capacidad de una taxonomía para describir defectos. A continuación se describe cada atributo del marco. El nombre original de

aquellos atributos que derivan de la propuesta de Freimut, aparecen entre paréntesis al final de la descripción. Si el atributo no se deriva de la propuesta de Freimut, aparece la palabra “nuevo” al final de la descripción.

Ubicación sospechada Se refiere al lugar donde se sospecha que se encuentra el defecto. Este atributo se registra al momento en que se produce la falla. Puede referirse a la actividad responsable de construir el producto que contiene el defecto, o a un producto propiamente dicho. (Ubicación)

Ubicación real Se refiere al lugar donde se encuentra el defecto realmente. Puede referirse a la actividad responsable de construir el producto que contiene el defecto, o a un producto propiamente dicho. (Ubicación)

Fase de provocación de la falla Se refiere a la fase en la que se encuentra el proyecto en el momento en que se provoca la falla. (Tiempo)

Fase de inyección Se refiere a la fase en la que se encuentra el proyecto en el momento en que se inyecta el defecto. (Tiempo)

Síntoma Se refiere a qué se observa, es decir, lo que puede verse cuando se provoca la falla. (Síntoma)

Tipo Se refiere al tipo de defecto que se detectó. (Nuevo)

Mecanismo de provocación de falla Se refiere a cómo se provocó la falla, mediante qué actividad. (Mecanismo)

Mecanismo de inyección Se refiere a cómo se inyectó el defecto. (Mecanismo)

Impacto de la falla Se refiere al impacto que produce la falla en el producto, al momento en que ocurre la falla. (Resultado final)

Impacto del defecto Se refiere al impacto que produce el defecto en el producto, luego de detectar el mismo. (Resultado final)

Causa (Error) Se refiere al error humano que causó la inyección del defecto. (Causa)

Fuente responsable Se refiere a dónde se desarrolló el producto en el cual se inyectó el defecto. Es un atributo que representa, por ejemplo, si el defecto se encuentra en un producto desarrollado en el proyecto actual, o si se encontró en un producto desarrollado en casa pero no especialmente para este proyecto o si el mismo se encuentra en un producto externo. (Nuevo)

Recorrección Se refiere a si el defecto fue introducido al corregir un defecto anterior. (Nuevo)

Probabilidad de ocurrencia Se refiere a la probabilidad estimada de ocurrencia de la falla. Este atributo se registra al momento en que se produce la falla. (Nuevo)

Gravedad Se refiere a qué tan grave es la falla que ocurrió. (Gravedad)

Prioridad Se refiere a qué tan rápido debe ser corregido el defecto. (Nuevo)

Costo de detección Se refiere al tiempo consumido en la detección del defecto, luego de la provocación de la falla. (Costo)

Costo estimado de corrección Se refiere al tiempo que se estima que llevará su corrección, luego de la detección del defecto. (Costo)

Costo de corrección real Se refiere al tiempo consumido en la corrección del defecto. (Costo)

Así como se incluyeron como atributos la fase de provocación de la falla y la fase de inyección del defecto, se pensó en incluir atributos que representen la fase de detección y la fase de corrección del detecto. Se decidió no incluir estos atributos en el marco, ya que no se considera que puedan aportar conocimiento útil para analizar. Consideramos que es una decisión independiente del defecto el elegir en que fase detectar el defecto. Es decir, ya se sabe que surgió una falla, pero la decisión de en qué momento detectar el defecto, es una decisión particular

de cada proyecto. Análogo es el razonamiento para la fase de corrección del defecto, luego de que el mismo fue detectado.

La vista basada en Propiedades, se basa en las propiedades deseables propuestas por Freimut, incluyendo también propiedades que entendemos necesario y útil evaluar en las distintas taxonomías. A continuación se describen las nuevas propiedades propuestas por nosotros y el posible conjunto de valores para cada una.

Calidad de la descripción de valores de atributos Refiere a expresar qué nivel de calidad tiene la descripción de los valores de atributos. Resulta interesante conocer el nivel de calidad, ya que sólo indicar la existencia o no de una descripción no brinda un panorama de qué tanto aporta dicha descripción a la taxonomía. Para esta propiedad no se definen valores posibles, sino que para cada taxonomía se debe expresar el nivel de calidad existente.

Calidad de ejemplos de valores de atributos Se refiere a expresar qué nivel de calidad tienen los ejemplos de los valores de atributos. Resulta interesante conocer el nivel de calidad de los ejemplos, ya que sólo indicar la existencia o no de los mismos, no brinda un panorama de qué tan claro es el valor de un atributo y cuándo dicho valor aplica correctamente. Los valores posibles para esta propiedad son "Buena", "Mala" o "No contiene ejemplos", según la taxonomía contenga ejemplos claros y concisos, ambiguos o no contenga ejemplos, respectivamente.

Clasificación Se refiere a cómo se clasifica la taxonomía según [12]. Los valores posibles para esta propiedad son "Taxonomía de defectos", "*Root cause analysis*" o "Clasificación de defectos".

Estructura Se refiere a cuál es la estructura de la taxonomía. Resulta interesante conocer la estructura de una taxonomía para tener una visión clara de la relación existente entre sus atributos. Los valores posibles para esta propiedad son "Jerárquica", "Árbol", "Ortogonal", "Semi-Ortogonal" o "Simple". Es deseable que la estructura sea ortogonal, ya que eso asegura que aspectos diferentes de un defecto sean capturados en atributos diferentes.

Nivel de generalidad Se refiere a la capacidad de aplicación de la taxonomía a distintos proyectos o procesos de software. Resulta de interés conocer el nivel de generalidad para conocer cuando se puede utilizar la taxonomía. Los valores posibles para esta propiedad son "Específica" (sólo se puede aplicar para un tipo software determinado), "Intermedia" (sólo se puede aplicar en una etapa del desarrollo de software) o "Genérica" (aplicable en cualquier momento del ciclo de vida del desarrollo).

Adaptabilidad Se refiere a la capacidad de expansión y modificación de la taxonomía acorde a las necesidades. Resulta de interés conocer la capacidad de adaptación de una taxonomía ya que es importante conocer si se puede modificar o extender la taxonomía existente para lograr un objetivo particular o si es necesario crear algo totalmente nuevo para ello. Los valores posibles para esta propiedad son "Adaptable", "Medianamente adaptable" o "No adaptable".

Tiempo de aprendizaje Se refiere al tiempo que se demora en aprender a usar una taxonomía. Resulta de interés conocer el costo medio de aprendizaje, ya que en el caso que el tiempo sea una limitante al momento de decidirse por una taxonomía u otra, es importante tener una estimación de cuanto tiempo implica su estudio. Los valores posibles para esta propiedad son "Alto", "Medio" o "Bajo", según el tiempo que consuma su aprendizaje.

Tiempo de clasificación Se refiere al tiempo que se demora en clasificar un defecto en una taxonomía luego de saber usarla. Al igual que la propiedad anterior, el interés surge porque el tiempo puede ser una limitante al momento de elegir la taxonomía a incorporar en un proyecto. Los valores posibles para esta propiedad son "Alto", "Medio" o "Bajo", según el tiempo que consuma su aplicación.

A continuación se lista las propiedades propuestas por Freimut en la sección 2.3 y el posible conjunto de valores para cada una.

Valores de atributo mutuo-excluyentes El cumplimiento de esta propiedad implica que al momento de clasificar un atributo para un defecto, sólo un valor es correcto. Los valores posibles para esta propiedad son "Sí" o "No", según la taxonomía cumpla o no con dicha propiedad.

Completitud en valores de atributos El cumplimiento de esta propiedad implica que al momento de clasificar un atributo para un defecto, siempre exista al menos un valor correcto. Los valores posibles para esta propiedad son “Sí”, “Aparente” o “No”, según la taxonomía cumpla, aparentemente cumpla o no cumpla con dicha propiedad.

4.2. Comparación

Atributos	HP	Kaner	Binder
Ubicación sospechada			
Ubicación real	Origen		Origen
Fase de provocación de la falla			
Fase de inyección			
Síntoma			
Tipo	Tipo	Tipo	Tipo
Mecanismo de provocación de la falla			
Mecanismo de inyección	Modo		Tipo (no siempre contemplado)
Impacto de la falla			
Impacto del defecto			
Causa (Error)			
Fuente responsable			
Recorreción			
Probabilidad de ocurrencia			
Gravedad			
Prioridad			
Costo de detección			
Costo estimado de corrección			
Costo de corrección real			

Cuadro 2. Comparación de taxonomías HP, Kaner y Binder contra el marco teórico, vista Atributos

La forma en que la vista basada en atributos debe ser utilizada es la siguiente: para cada atributo en la vista, buscar un atributo correspondiente en la taxonomía que se está evaluando. En los Cuadros 2 y 3, se evalúan las seis taxonomías contra el marco teórico, vista Atributos. En dichos Cuadros se puede observar, para cada taxonomía, qué atributo se corresponde con qué atributo del marco teórico.

Se observa que el único atributo que se encuentra en todas las taxonomías es el tipo de defecto.

Binder, Kaner, Beizer y HP clasifican el defecto solamente cuando el mismo ya fue detectado. ODC e IEEE clasifican al momento en que se provoca la falla y también luego de detectar el defecto. IEEE además clasifica al momento en que se corrige y al finalizar el registro y seguimiento del defecto.

Se puede observar que existen atributos planteados en el marco teórico que no están contemplados en ninguna de las taxonomías vistas. A continuación se lista cada uno y se detalla cuál es el motivo de su inclusión en el marco teórico.

Fase de inyección Resulta interesante registrar la fase en que se inyecta el defecto, ya que dicho conocimiento puede utilizarse para prevenir la inyección de defectos similares en proyectos futuros.

Atributos	IEEE	ODC	Beizer
Ubicación sospechada	Causa sospechada		
Ubicación real	Fuente, Causa actual	Target	
Fase de provocación de la falla	Fase del Proyecto		
Fase de inyección			
Síntoma	Síntoma, Estado del producto		
Tipo	Tipo	Tipo	Tipo
Mecanismo de provocación de la falla	Actividad del proyecto	Trigger, Actividad	
Mecanismo de inyección		Calificador	
Impacto de la falla	Valor para el cliente, Misión y seguridad	Impacto	
Impacto del defecto	Costo del proyecto, Calidad y confiabilidad del proyecto, Riesgo del Proyecto, Sociedad		
Causa (Error)			
Fuente responsable		Fuente, Edad	
Recorrección		Edad	
Probabilidad de ocurrencia	Repetibilidad		
Gravedad	Gravedad		
Prioridad	Prioridad		
Costo de detección			
Costo estimado de corrección	Cronograma del Proyecto		
Costo de corrección real			

Cuadro 3. Comparación de taxonomías IEEE, ODC y Beizer contra el marco teórico, vista Atributos

Causa (Error) Resulta interesante registrar cuál fue el error humano que provocó la inyección del defecto, ya que ser concientes de ello puede ayudar en la prevención de defectos en futuros proyectos. Algunos resultados de *root cause analysis* dan posibles valores para este atributo. Por ejemplo en [10], se plantea un atributo Causa con los valores Educación, Supervisión, Comunicación, Herramientas y Transcripción. En [9] se propone un atributo que captura distintos tipos de causas: Causas Humanas (falta de conocimiento, problemas de comunicación), Causas del Proyecto (falta de tiempo, error de gestión) o Causas de Revisión (no se hizo inspección o fue incompleta, participación inadecuada).

Costo de detección Resulta interesante ya que se podría realizar un análisis de costos según los tipos de defectos y ver si existe alguna relación que luego permita estimar los cambios en el cronograma del proyecto. Además, conocer qué tipos de defectos son costosos de detectar puede colaborar en la mejora del proceso grupal e individual, para evitar los defectos más costosos.

Costo de corrección real Al igual que el costo de detección, resulta interesante su registro para un posterior análisis de costos, con el objetivo de estimar correctamente los cambios en el cronograma del proyecto. También, conocer qué tipos de defectos son costosos de corregir puede colaborar en la mejora de los procesos.

También se puede observar que los atributos Resolución, Acción Correctiva y Disposición de la taxonomía propuestos por la IEEE, no se encuentran contemplados en ningún atributo del marco. No se incorporaron los significados de Resolución y Acción Correctiva, ya que para nosotros la resolución y acción a tomar para prevenir un defecto no surge de un único defecto, sino que son acciones que se toman a partir de la información reelevada de un conjunto de defectos. Tampoco se incorporó el significado de Disposición ya que el mismo tiene como objetivo indicar la etapa final del registro y seguimiento de un defecto. No consideramos que sea una característica a reelevar de un defecto.

Propiedades	**HP**	**Kaner**	**Binder**
Valores de atributo mutuo-excluyentes	Si	Si	No
Completitud en valores de atributos	Aparente	Aparente	Aparente
Calidad de la descripción de valores de atributos	Buena: descripciones claras, sin ambiguedades	Mala: no todos los valores están explicados	No se describen los valores
Calidad de ejemplos de valores de atributos	Buena	No contiene ejemplos	No contiene ejemplos
Clasificación	Clasificación de defectos	Taxonomía de defectos	Taxonomía de defectos
Estructura	Semi-Ortogonal	Jerárquica	Simple
Nivel de generalidad	Genérica	Genérica	Específica
Adaptabilidad	Adaptable	Adaptable	Adaptable
Tiempo de aprendizaje	Bajo	Bajo	Alto
Tiempo de clasificación	Bajo	Bajo	Alto

Cuadro 4. Comparación de taxonomías HP, Kaner y Binder contra el marco teórico, vista Propiedades

En los Cuadros 4 y 5, se presenta la comparación de las seis taxonomías vistas contra el marco teórico, vista Propiedades.

Respecto a los valores para un atributo, las taxonomías de HP, Kaner, IEEE y ODC presentan la propiedad de conformar un conjunto mutuo-excluyente. Esto significa que nunca dos o más valores de un mismo atributo se corresponden con un mismo defecto. Sin embargo, en las taxonomías de Binder y Beizer, el conjunto de valores para un atributo no presenta esa propiedad, lo que puede generar datos inconsistentes ya que se podría clasificar un mismo defecto de distintas maneras.

Observando la estructura de las taxonomías, observamos que ODC e IEEE son ortogonales. IEEE además es jerárquica al igual que las taxonomías de Beizer, Binder y Kaner, pues en las cuatro taxonomías el atributo tipo de defecto es refinado en varios subniveles. La taxonomía de Binder también es arborescente, ya que el valor del atributo Tipo depende del origen seleccionado previamente. La taxonomía de HP es semi-ortogonal, ya que el valor del atributo Tipo depende del valor elegido para el atributo Origen, pero el atributo Modo es independiente de los otros atributos.

La completitud en los valores de los atributos es una propiedad difícil de demostrar, ya que se debería tomar una muestra de defectos y clasificarlos en cada taxonomía para verificar que realmente todos los defectos pueden ser clasificados en los atributos correspondientes. Por eso es que la única taxonomía que realmente podemos

Propiedades	IEEE	ODC	Beizer
Valores de atributo mutuo-excluyentes	Si	Si	No
Completitud en valores de atributos	Aparente	Aparente	Si
Calidad de la descripción de valores de atributos	Buena: descripciones claras, sin ambiguedades	Buena: descripciones claras, sin ambiguedades	Mala: descripción superficial, ambigua en algunos casos
Calidad de ejemplos de valores de atributos	Buena	Buena	Mala
Clasificación	Clasificación de defectos	Clasificación de defectos	Taxonomía de defectos
Estructura	Ortogonal y Jerárquica	Ortogonal	Jerárquica
Nivel de generalidad	Genérica	Genérica	Genérica
Adaptabilidad	Adaptable	Adaptable	Adaptable
Tiempo de aprendizaje	Alto	Bajo	Alto
Tiempo de clasificación	Alto	Bajo	Alto

Cuadro 5. Comparación de taxonomías IEEE, ODC y Beizer contra el marco teórico, vista Propiedades

afirmar que cumple con esta propiedad es Beizer, ya que para su único atributo existe el valor “Otros”, donde allí clasifican los defectos que no se correspondían con ninguno de los valores anteriores. Las demás taxonomías aparentan cumplir con la completitud, pero no es posible afirmarlo.

Respecto a la calidad de las descripciones y ejemplos de los valores de un atributo, se aprecia una variación de la misma en las distintas taxonomías. De todas formas, podemos decir que las taxonomías de HP, IEEE y ODC presentan buena calidad, ya que las mismas poseen descripciones completas, que no dan lugar a confusión, además de incluir ejemplos concisos. Por otro lado, decimos que las taxonomías de Kaner, Binder y Beizer tienen una mala calidad, ya que o no contienen descripciones ni ejemplos, o dichos atributos son superficiales, ambiguos o incompletos.

Respecto a los tiempos de aprendizaje, se agrupó con el valor “Bajo” aquellas taxonomías que se consideran fácilmente comprensibles, ya sea por la claridad y completitud de su presentación y/o por ser taxonomías que comprenden una cantidad de atributos pequeña. A las taxonomías presentadas por Binder y Beizer, se les puso el valor “Alto” ya que si bien sólo están compuestas por un único atributo, existen muchos valores posibles para ese atributo. Eso sumado a descripciones vagas y una mala calidad de los ejemplos, resulta en un tiempo de aprendizaje mucho mayor al del otro grupo. La taxonomía presentada por la IEEE también se considera con tiempo de aprendizaje “Alto”, ya que contiene un conjunto muy amplio de atributos que deben ser estudiados y entendidos.

Análogo es el razonamiento propuesto para los tiempos de clasificación. A aquellas taxonomías con descripciones sin ambiguedades, buenos ejemplos y/o con pocos atributos a clasificar, se las agrupó con el valor “Bajo”. Las taxonomías con mayor cantidad de atributos o aquellas que no están claramente definidas ni ejemplificadas, donde el usuario generalmente demora su clasificación por dudar entre dos o más valores, se considera que tienen un tiempo de clasificación “Alto”.

Clasificar un tiempo como “Alto”, “Medio” o “Bajo” no es la mejor forma de hacerlo, ya que puede ser demasiado subjetiva y poco precisa. A futuro se podrían calcular los tiempos medios de aprendizaje y clasificación para las diversas taxonomías, y poner dichos valores para poder realizar una comparación más exacta y representativa de la realidad.

Se observa que la cantidad de atributos, la calidad de las descripciones de los valores de atributos y la calidad de los ejemplos presentados en cada taxonomía influyen significativamente en el tiempo de aprendizaje y clasificación de la taxonomía.

5. Conclusiones

Presentamos la meta taxonomía de Freimut, donde se analizan posibles atributos y propiedades de taxonomías, sus distintas estructuras y se expone una guía de cómo incorporar una taxonomía a un proceso.

Presentamos un conjunto variado de taxonomías de defectos de software, las cuales brindan un panorama completo de las taxonomías existentes tanto en la literatura como en la industria.

Desarrollamos y presentamos un marco de comparación original para taxonomías de defectos. Este marco extiende y mejora las ideas de Freimut. El mismo está compuesto por dos vistas relevantes: atributos y propiedades. La primer vista es útil para evaluar las características de defectos que son consideradas para una taxonomía dada. La segunda vista es útil para evaluar las cualidades y las restricciones de los atributos.

Utilizando el marco evaluamos y comparamos todas las taxonomías presentadas. Los resultados muestran que cada taxonomía considera distintas caracteristicas de un defecto. Algunas de ellas son más completas que otras, desde el punto de vista de atributos. Sin embargo, aquellas menos completas tienen un conjunto más amplio de valores para sus atributos, por ejemplo, Beizer y Binder. El análisis también muestra que hay diferencias en las propiedades entre las taxonomías.

Como trabajo a futuro, es importante: analizar el impacto de tener o no un atributo específico del marco en una taxonomía, analizar los valores propuestos en cada taxonomía para cada atributo, analizar si distintos niveles de pruebas (unitaria, integración, sistema) requieren distintas taxonomías y finalmente analizar la forma en que las taxonomías han sido utilizadas en la industria. Actualmente estamos trabajando en estos dos últimos puntos.

Referencias

[1] B. Beizer. *Software Testing Techniques, Second Edition*. Van Nostrand Reinhold Co., 1990. 2.2, 3.6

[2] R. V. Binder. *Testing Object-oriented Systems Models, Patterns, and Tools*. Addison-Wesley, 1999. 3.3

[3] R. Chillarege. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, McGraw-Hill Book Company, 1996. 2.2, 3.5

[4] B. Freimut. Developing and using defect classification schemes. Technical report, Fraunhofer IESE, 2001. 2, 4

[5] R. B. Grady. *Practical Software Metrics For Project Management and Process Improvement*. Hewlett-Packard, 1992. 2.2, 3.1

[6] IEEE. *IEEE 1044-1993 Standard Classification for Software Anomalies*. Institute of Electrical and Electronics Engineers, 1993. 3.4

[7] IEEE. *IEEE Guide to Classification for Software Anomalies*. Institute of Electrical and Electronics Engineers, 1995. 2.4

[8] C. Kaner, J. Falk, and H. Q. Nguyen. *Testing Computer Software (2nd. Edition)*. International Thomson Computer Press, 1999. 3.2

[9] M. Leszak, D. E. Perry, and D. Stoll. A case study in root cause defect analysis. *Proceedings of the 22nd international conference on Software engineering*, 2000. 2.5, 4.2

[10] R. G. Mays, C. L. Jones, G. J. Holloway, and D. Studinski. Experiences with defect prevention. *IBM SYSTEMS JOURNAL*, 1990. 2.5, 4.2

[11] D. Vallespir and S. D. León. Análisis y ejemplos de la taxonomía de defectos de beizer. Technical Report RT 08-19, Instituto de Computación – Facultad de Ingeniería - Universidad de la República, 2008. 3.6

[12] S. Wagner. *Defect Classifications and Defect Types Revisited*. Technische Universität München, 2008. 2.5, 4.1