# Universidad de las Ciencias Informáticas

# Facultad 5


# Título: MÓDULO DE SIMULACIÓN DE MONTECARLO PARA EL PROYECTO "SISTEMA INTEGRAL DE CONFIABILIDAD"

**Trabajo de Diploma para optar por el título de**

**Ingeniero en Ciencias Informáticas**

**Autor: Yoelvis Mulen Llorente**

**Tutor: Ing. Riolvi Acosta Gonzalez**

**La Habana**

**Junio de 2012**

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas el derecho patrimonial de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yoelvis Mulen Llorente

_____

Firma del Autor

Ing. Riolvi Acosta Gonzalez

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Riolvi Acosta Gonzalez.

Edad: 25 años.

Ciudadanía: Cubana.

Institución**:** Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias de la Informática.

Categoría Docente: Trabajador no Docente.

E-mail: riolvis@uci.cu

AGRADECIMIENTOS

Le agradezco a Jesucristo mi salvador que me tomó de su mano y me ha hecho la persona que soy hoy, gracias por ser mi amigo y por tu amor incondicional hacia toda la humanidad.

A mi novia que ha sido durante estos cinco años la persona que ha estado a mi lado dándome su amor, agradezco haberte conocido y espero pasar mi vida a tu lado.

A mi mamá (Biti) que siempre me ha enseñado a seguir adelante, a no conformarme con una nota que no sea 5, quién me ha dado ánimo cuando yo he sido pesimista, por el esfuerzo que ha tenido que hacer porque yo pueda graduarme de esta universidad, por confiar en mí cuando ni yo mismo confiaba, gracias por todo tu amor y dedicación.

A mi abuela (Lima) por el esfuerzo que ha hecho para criarme, dándome todo el amor del mundo, quitándose muchas veces su comida para dármela, ella que junto a mi mamá han hecho el papel de madre y padre, gracias por todo.

A mi hermanito que lo quiero mucho, y me ha soportado todas mis torturas y humillaciones.

A mis tíos Ricardo y Regla por ayudarme en todo lo que he necesitado y por el cariño que me han dado.

A mis tíos Frank y Sandra que aunque se encuentren lejos nos hemos podido comunicar por el chat, y me han dado sus consejos y ayuda.

A mis primos Ronald y Freddy que hay sido como mis hermanos desde la infancia.

Al resto de mi familia y a mis vecinos que también han aportado su granito de arena para hoy poder estar donde estoy.

A mis hermanos de la fe que he podido disfrutar de su compañía, consejos y amor.

A mi tutor por apoyarme durante la realización de este trabajo, quién ha sido como mi compañero de tesis.

A mis amigos, compañeros de grupo y apartamento y todos aquellos con quién he compartido estos años.

A todos gracias por haber sido parte de estos 5 años de estudio y sacrificio, pero también de amistad, risas, amor, fútbol, cultos poderosos y mucho más. Gracias.

DEDICATORIA

Dedico este trabajo a mi mamá Bárbara Llorente González y a mi abuela Evangelina González González.

RESUMEN

La presente investigación se centra en el desarrollo de un módulo de simulación de Montecarlo para el proyecto SIC (Sistema Integral de Confiabilidad), el cual permitirá la caracterización probabilística de variables aleatorias, la propagación de la incertidumbre asociada a cada variable de los modelos matemáticos manejados en cada una de sus metodologías y cuantificar el nivel de incertidumbre asociada a la salida de los mismos. Para la implementación de la solución se hizo un análisis de las bibliografías afines a la investigación y de las tecnologías y metodologías de software a utilizar. Como resultado se dotó a SIC de un módulo de código abierto, multiplataforma y bajo los principios de soberanía que le permite realizar la simulación de Montecarlo.

PALABRAS CLAVE

Caracterización probabilística, modelos matemáticos, propagación de la incertidumbre, simulación de Montecarlo, variables aleatorias.

# Índice de Contenido.

# INTRODUCCIÓN

La dinámica de las empresas ha llevado a comprender que la gestión eficaz de los activos físicos constituye una tarea de suma importancia para garantizar la sostenibilidad y calidad de vida de la organización. Además de ser la fuente de grandes ventajas competitivas, pero a su vez también un área de extremo cuidado.

Ante esta panorámica se han ido desarrollando diversas estrategias para la gestión de los activos, con el objetivo de optimizar el costo en el ciclo de vida de los mismos. Entre las estrategias de gestión, se destaca la confiabilidad operacional como la de mayor ímpetu, pues persigue la mejora continua de procesos, incorporando de manera sistemática avanzadas herramientas de diagnóstico, técnicas de análisis y nuevas tecnologías, para optimizar la gestión, planeación, ejecución y control de la producción industrial.

Actualmente en Cuba se realizan enormes esfuerzos para convertirse en uno de los mayores productores de software, siendo este uno de los motivos por el que existe la Universidad de las Ciencias Informáticas (UCI). La UCI se divide en 7 facultades a las cuales responden varios centros productivos de desarrollo de software. En la facultad 5 se encuentra el Centro de Informática Industrial (CEDIN), el cual se dedica a la automatización de procesos industriales, dentro de sus numerosos proyectos productivos se encuentra el desarrollo del proyecto Sistema Integral de Confiabilidad (SIC). El proyecto SIC está conformado por un equipo multidisciplinario de profesores, ingenieros y estudiantes, y tiene como objetivo fundamental desarrollar un sistema integral de código abierto, multiplataforma y que cumpla con los principios de soberanía tecnológica del CEDIN, el cual incluirá diversas aplicaciones orientadas a la confiabilidad operacional.

En las empresas se toman una gran cantidad de decisiones que involucran variables aleatorias, por lo que se hace necesario el uso de algún modelo matemático para la evaluación del nivel de incertidumbre de las mismas, de manera que se puedan realizar predicciones con cierto nivel de certeza. En el área de la confiabilidad operacional estos modelos se manejan matemáticamente mediante el método numérico de simulación de Montecarlo, el cual permite determinar las distribuciones probabilísticas que mejor se ajustan a una serie de datos específicos, cuantificando así el nivel de incertidumbre asociado a las variables de salida de dichos modelos.

Para la toma de decisiones los ingenieros en confiabilidad utilizan varias herramientas de código cerrado, con altos costos de licenciamiento, dependientes de plataforma Microsoft Windows y/o complementos de hojas de cálculo Excel. Estas herramientas por si solas brindan información incompleta del proceso de simulación de Montecarlo, lo que propicia que se maneje manualmente gran cantidad de datos de forma desorganizada e incompleta. Provocando retrasos en la toma de decisiones en los ingenieros en confiabilidad en sus empresas.

Actualmente el proyecto SIC no posee un módulo de código abierto, multiplataforma y bajo los principios de la soberanía tecnológica, que permita, realizar pruebas de bondad de ajuste para determinar la distribución de probabilidad que mejor representa a las múltiples variables que maneja, propagar la incertidumbre asociada a cada variable de los modelos matemáticos manejados en cada una de sus metodologías y cuantificar el nivel de incertidumbre asociada a la salida de los mismos, mediante la aplicación del método de simulación de Montecarlo.

Analizando la situación problémica expuesta con anterioridad, se define como **problema a resolver**:

¿Cómo automatizar la simulación de Montecarlo en el proyecto SIC?

Teniendo en cuenta el problema a resolver, se precisa como **objeto de estudio**: La simulación de Montecarlo.

Por todo lo anterior y para darle solución al problema a resolver, se plantea el siguiente **objetivo**: Desarrollar un módulo de tratamiento de variables aleatorias utilizando la simulación de Montecarlo para el Sistema Integral de Confiabilidad.

Por tanto, el **campo de acción** es: La simulación de Montecarlo aplicada a la confiabilidad operacional.

Para dar cumplimiento al objetivo general se plantean las siguientes **tareas investigativas**:

1. Elaboración del marco teórico centrado en la simulación probabilística mediante el Método de simulación de Montecarlo en la confiabilidad operacional.
2. Análisis y diseño del módulo de simulación de Montecarlo.
3. Implementación del módulo de simulación de Montecarlo.
4. Integración del módulo de simulación de Montecarlo a la aplicación del Sistema Integral de Confiabilidad.
5. Documentación y estandarización del código fuente del módulo de simulación de Montecarlo.

6. Diseño y desarrollo de pruebas unitarias al módulo de simulación de Montecarlo.

De acuerdo con el problema científico planteado se define la siguiente **idea a defender**:

El desarrollo del módulo de simulación de Montecarlo para el proyecto SIC automatizará la caracterización probabilística de variables aleatorias y la propagación de la incertidumbre asociada a cada variable aleatoria en los modelos matemáticos que maneja.

Durante toda la investigación se emplean varios métodos científicos en la búsqueda y procesamiento de la información:

A nivel teórico:

- Análisis-síntesis e inducción-deducción: se utiliza para el análisis, estudio y recopilación de los conceptos, teorías, métodos y fórmulas referentes al tema de investigación con el objetivo de desarrollar un módulo de tratamiento de variables aleatorias utilizando la simulación de Montecarlo.
- Análisis histórico-lógico: se utiliza en el análisis de los antecedentes y tendencias actuales de las herramientas y tecnologías existentes referentes al tema de investigación.

A nivel empírico:

- Análisis de las fuentes de información: se usa para consultar fuentes de información relacionadas con la investigación.

El presente documento se encuentra estructurado por tres capítulos, con toda la información referente al tema de investigación. Estos se encuentran distribuidos de la siguiente manera:

Capítulo I: “Fundamentación Teórica”, se realiza un estudio de los conceptos asociados al tema de la investigación, la metodología y las tecnologías a utilizar.

Capítulo II: “Análisis y Diseño del Sistema”, se realiza la especificación de los requisitos funcionales y no funcionales, se describe los casos de uso del sistema y sus actores, se analiza la arquitectura de la solución, el diseño y descripción de los subsistemas y clases más importantes para la comprensión del código fuente.

Capítulo III: “Implementación y Pruebas”, se describe el modelo de implementación, los requerimientos para el despliegue y el diseño e implementación de pruebas para validar la solución.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## 1.1. Introducción.

El presente capítulo aborda el estado del arte referente a la caracterización probabilística de variables aleatorias y a la propagación de la incertidumbre asociada a cada variable de los modelos matemáticos en los procesos de confiabilidad operacional mediante la simulación de Montecarlo. Se tratan los principales conceptos asociados al tema, así como las tecnologías y metodologías de desarrollo de software a utilizar en la implementación de la solución.

## 1.2. Probabilidad.

La **probabilidad** es la medición numérica (entre 0 y 1) de la posibilidad de ocurrencia de un evento (1). Formalmente el término de "probabilidad" ha sido definido por dos escuelas de pensamiento que regulan el significado y en consecuencia la aplicación de la probabilidad. Estas escuelas son conocidas como: Escuela Frecuentista o Clásica de Probabilidad y la Escuela Subjetivista o Bayesiana de Probabilidad (ver Ilustración 1).

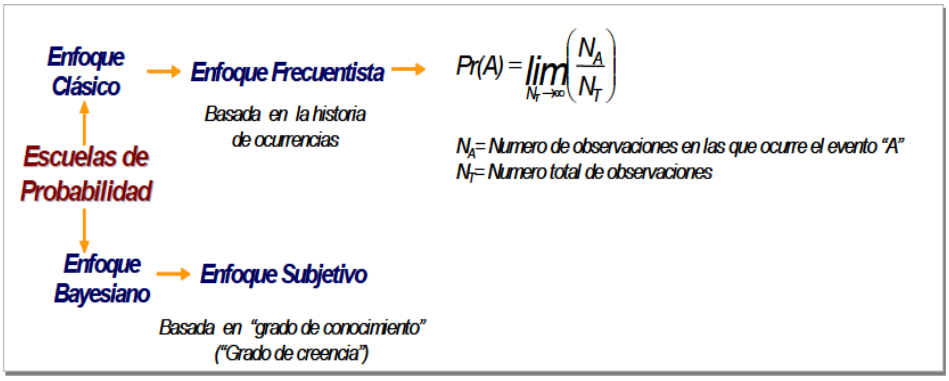


Ilustración 1. Probabilidad.

**Definición Frecuentista o Clásica:** la frecuencia de ocurrencia de un evento es un indicador de probabilidad de dicho evento, es decir, si el evento es muy frecuente se asume que su probabilidad de ocurrencia es alta (su probabilidad tiende a 1); si por el contrario, el evento es poco frecuente se asume que su probabilidad de ocurrencia es baja (su probabilidad tiende a 0). (1)

**Definición Subjetivista o Bayesiana:** grado de confianza en la veracidad de una proposición (hipótesis que puede ser probada como verdadera o falsa).

### 1.3. Variable aleatoria.

La variable aleatoria es una variable x que por sus características pueda tomar un conjunto de valores (x1, x2, x3, x4,... xn-1), cada uno de los cuales tiene una probabilidad de ocurrencia (p1, p2, p3, p4,... pn-1), sin que se pueda asegurar específicamente cuál de todos estos probables valores tomará la variable. (1)

Las variables aleatorias pueden ser discretas o continuas. Las variables aleatorias discretas son aquellas que solo pueden tomar un número finito o infinito de valores numerables o contables, mientras que las variables aleatorias continuas pueden tomar todos los valores de un intervalo dado.

### 1.4. Distribuciones de Probabilidad.

Las **distribuciones de probabilidad** son modelos que describen la forma en que se espera que varíen los resultados o probables valores de una variable aleatoria. Estas se caracterizan por tres criterios fundamentales: el valor central o medida de posición (*la media, la mediana o la moda*), el grado de dispersión (*la desviación estándar*) y la forma de la curva (*forma general de la distribución probabilística*). (2)

Las distribuciones de probabilidad se pueden clasificar en paramétricas y no paramétricas. Las **distribuciones de probabilidad paramétricas** son funciones matemáticas teóricas que describen la forma en que se espera que varíen los resultados de un experimento, es decir, funciones matemáticas que relacionan los diversos valores que pueden tomar una variable aleatoria con la probabilidad de ocurrencia de cada uno de ellos. Mientras que las **distribuciones de probabilidad no paramétricas** son modelos gráficos que representan un grupo particular de observaciones de una variable aleatoria y que relacionan los diversos valores de la variable que se analiza, con su probabilidad de ocurrencia (ver Ilustración 2).

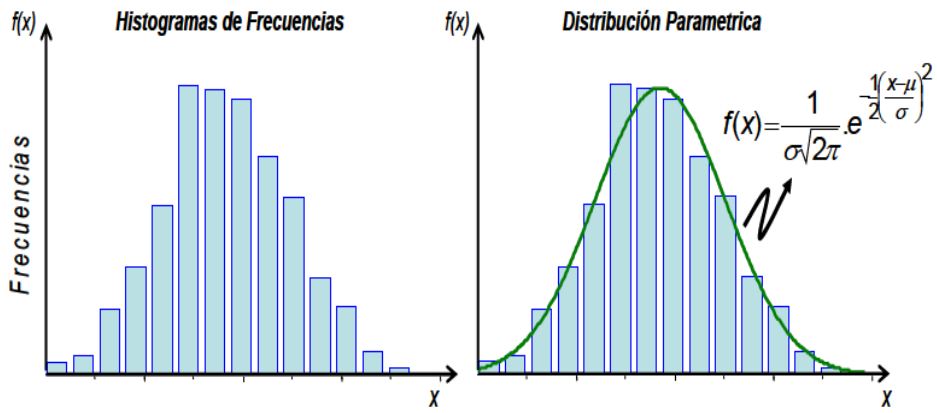


Ilustración 2. Distribuciones de Probabilidad.

Los negocios en el área de la Confiabilidad Integral se encuentran en la expectativa de la ocurrencia de eventos, sobre los cuales es necesario realizar algún tipo de inferencia para tomar decisiones en condiciones de incertidumbre. Por esta necesidad la investigación se basará en el estudio y análisis de las distribuciones de probabilidad paramétricas tanto para variables discretas como para variables continuas.

### 1.4.1. Distribuciones de Probabilidad Paramétricas para Variables Discretas.

**Función de densidad de probabilidades.**

Una variable aleatoria discreta X, que toma los valores x1, x2,..., xn con probabilidad de ocurrencia p(X=x1) = p1, p(X=x2) = p2, p(X=x3) = p3,..., p(X=xn) = pn, su función de probabilidad f(x) es la función que asigna a cada valor xi de la variable su correspondiente probabilidad pi; es decir, que f(x1)=p1, f(x2)=p2, F(x3)=p3,..., f(xn)=pn; en general f(xi) = pi (ver Ilustración 3). (1)

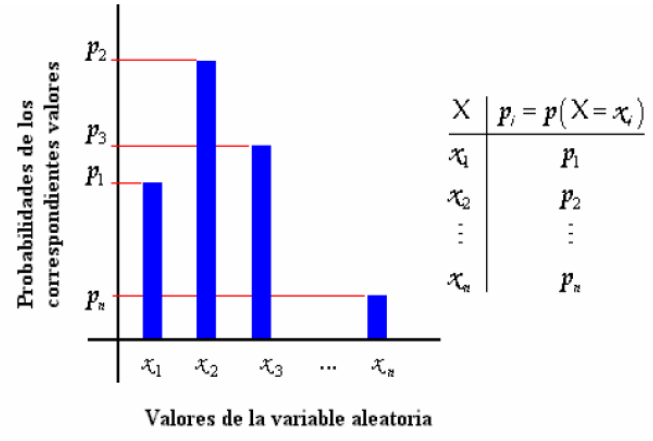


Ilustración 3. Distribuciones de Densidad de Probabilidad de Variables Discretas.

**Función de densidad acumulada.**

Es la probabilidad de que la acumulación de distintos valores de una variable aleatoria X tome valores menores o iguales que un cierto valor xi, como se aprecia en la Ilustración 4.

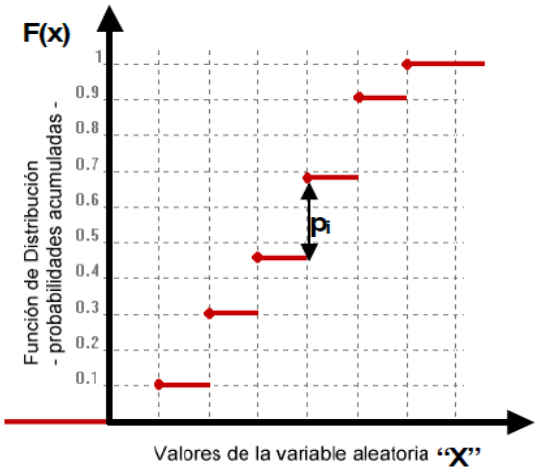


Ilustración 4. Distribuciones de Probabilidad Acumulada de Variables Discretas.

Ejemplo:

Para un valor $xj$ de la variable $X$, la probabilidad acumulada $F(xj)$ se interpreta como:

$F(xj) = p(X \leq xj)$ (Probabilidad de que $X$ tome valores menores o iguales a $xj$). Para calcular $F(xj)$, debe usarse la ecuación

$$F(xj) = \sum_{i=1}^{j} p_i$$

La sumatoria de todos los $p_i$ es igual a "1" $\Longrightarrow$

$$\sum_{i=1}^{n} p_i = 1$$

Algunas de las distribuciones de probabilidad paramétricas más utilizadas para variables discretas son:

- Distribución Binomial.
- Distribución Poisson.
- Distribución Hipergonométrica.
- Distribución Geométrica.
- Distribución Custom.

### 1.4.2. Distribuciones de Probabilidad Paramétricas para Variables Continuas.

**Función de densidad de probabilidades.**

Una función matemática f(x) es considerada una distribución de densidad de probabilidad de una variable aleatoria continua X si para cualquier intervalo de números reales [x1, x2] se cumple:

1. $f(x) \geq 0$
2. $\int_{-\infty}^{\infty} f(x)dx = 1$
3. $P(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} f(u)du$

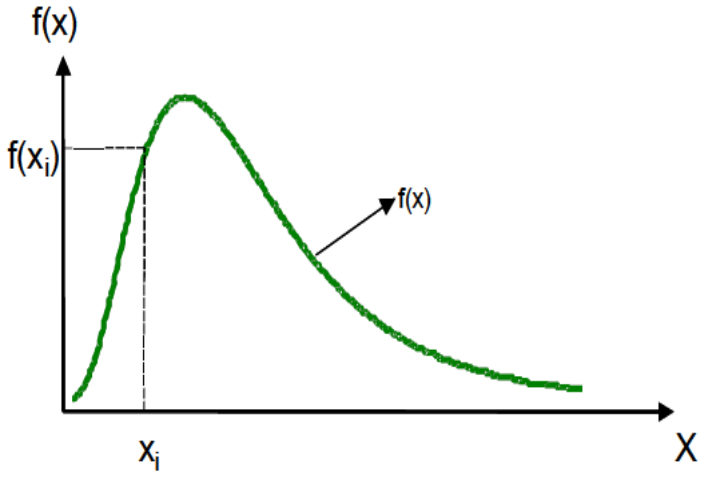


Ilustración 5. Distribuciones de Densidad de Probabilidad de Variables Continuas.

Esta función se puede deducir información muy importante sobre la variable aleatoria:

- El dominio de la variable aleatoria.
- El valor más probable de la variable aleatoria, que se ubicará debajo del punto más alto de la curva.
- El grado de dispersión de la variable.

**Función de densidad acumulada.**

Como se observa en la Ilustración 6, las funciones de densidad acumulada se distinguen por su común forma de "S" y relacionan cualquier valor $x_i$ de la variable aleatoria X, con la probabilidad de observar valores menores o iguales a dicho valor $x_i$.

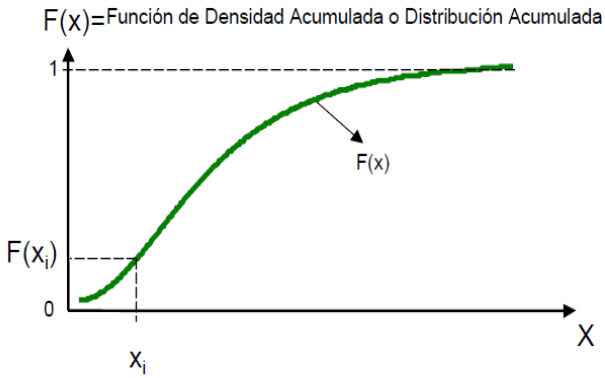

Ilustración 6. Distribuciones de Probabilidad Acumulada de Variables Continuas.

De manera que: $F(x_i) = p(X \leq x_i)$

Es muy importante notar que:

$0 \leq F(x) \leq 1$

$F(X)$ se construye acumulando las probabilidades obtenidas con la función $f(x) \Longrightarrow$

$$F(x_i) = \int_{-\infty}^{xi} f(x)dx$$

Cumpliéndose:

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

Algunas de las distribuciones de probabilidad paramétricas más utilizadas para variables continuas son:

- Distribución Normal.
- Distribución Lognormal.
- Distribución Exponencial.
- Distribución Weibull.
- Distribución Beta.
- Distribución Gamma.
- Distribución Triangular.
- Distribución Uniforme.

El estudio de los comportamientos de la gran cantidad distribuciones de probabilidad existente para representar la heterogeneidad y/o e incertidumbre de variables aleatorias ha posibilitado la clasificación en conjuntos o familias de distribución por diversos criterios (ver Ilustración 7).

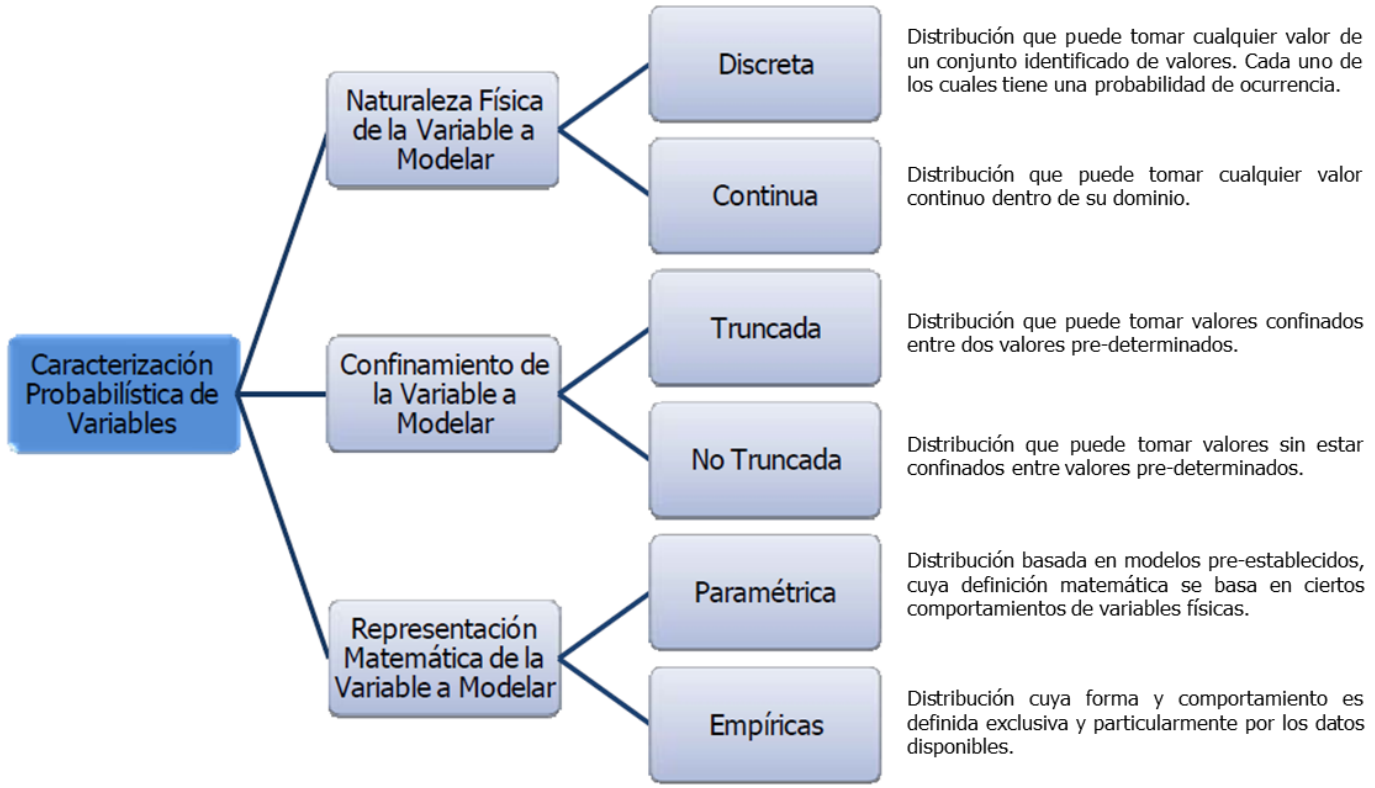


Ilustración 7. Clasificación de las distribuciones de probabilidad.

La investigación se centrará en el estudio de la caracterización probabilística de variables aleatorias por la naturaleza física de la misma, clasificación usada por los ingenieros en áreas de la confiabilidad para el manejo de sus procesos.

### 1.5. Pruebas de Bondad de Ajuste.

Las **pruebas de bondad de ajuste** permiten medir la discrepancia entre una función de distribución de probabilidad paramétrica y un conjunto de datos de una variable aleatoria (1). Una forma más sencilla de definirlas es la comparación entre la distribución paramétrica seleccionada y el histograma de frecuencias que se puede construir con los datos de la variable aleatoria.

Las pruebas de bondad de ajuste contemplan los siguientes pasos:

Paso 1. Seleccionar las distribuciones hipótesis y calcular los parámetros de cada una de ellas con los datos de la muestra.

Paso 2. Graficar cada una de las curvas de las distribuciones hipótesis teóricas obtenidas con los parámetros estimados en el paso anterior, con el histograma de los datos de la muestra.

Paso 3. Calcular para cada distribución hipótesis el valor llamado "*valor del prueba*" y compararlo contra el valor llamado "*valor crítico*".

Paso 4. Si el "*valor del prueba*" es menor que el "*valor crítico*" entonces la distribución hipotética se considera un buen ajuste y la hipótesis no es rechazada. Si por el contrario, el "*valor del prueba*" es mayor que el "*valor crítico*", la hipótesis se rechaza.

Paso 5. La distribución hipotética no rechazada que posea menor "*valor de prueba*" se considera como mejor ajuste al histograma de datos.

Algunas de las pruebas de bondad de ajuste que más utilizadas:

- Chi-Cuadrado.
- Kolmogorov-Smirnov.
- Anderson-Darling.
- WatsonG.
- WatsonU.

## 1.6. Caracterización Probabilística de Variables Aleatorias.

La caracterización probabilística de variables aleatorias es el proceso de seleccionar la distribución de probabilidad que más se ajuste a un conjunto de datos de una variable a modelar (1). Existen dos fuentes básicas de información para describir las variables, ya sea en función de la disponibilidad de datos provenientes de observaciones directas y/o medición de dispositivos de campo o en función de datos provenientes del conocimiento empírico (3) del proceso.

Los datos resultantes de mediciones de variables físicas de los procesos, observaciones directas hechas en campo y de la notificación automática o manual de ocurrencia de eventos; son caracterizados desde el punto de vista probabilístico. La cuantificación de la frecuencia de aparición de los datos en la muestra

posibilita hacer una evaluación probabilística objetiva y obtener la distribución de probabilidad que mejor se ajuste a dichos datos.

La obtención de los datos procedentes del conocimiento empírico del proceso se realiza de manera indirecta. Existen limitaciones que provocan que la recolección de datos sea de manera directa como pueden ser de tipo físico, refiriéndose a que exista mucha dificultad para medir la variable. Dada estas limitaciones se hace necesario acudir a la opinión de expertos lo cual facilita combinar el conocimiento de las personas sobre el proceso o área en análisis con los escasos datos que se disponen y la poca evidencia. Esta combinación puede traer como resultado nuevos valores por parte del experto producto de su experiencia en el área.

La caracterización probabilística de variables aleatorias con información de campo contempla los siguientes pasos:

Paso 1. Plantear las hipótesis de las distribuciones paramétricas que podrían hacer un buen ajuste.

Paso 2. Calcular los parámetros de cada una de las distribuciones hipótesis con los datos de la muestra.

Paso 3. Realizar alguna de las pruebas de bondad de ajuste.

### 1.7. Propagación de la incertidumbre.

La propagación de incertidumbre es el procedimiento mediante el cual se puede incluir y contabilizar la incertidumbre asociada a las variables de entrada, en un determinado modelo matemático (ecuación, inecuación o correlación), para cuantificar la incertidumbre de la variable de salida.

Es un proceso para resolver ecuaciones que tiene la característica de que las variables de entrada deben ser distribuciones de probabilidad. En caso de que las variables de entrada tengan incertidumbre, entonces la salida del modelo debe tener incertidumbre.

Para resolver este tipo de problemas, se dispone de diferentes herramientas matemáticas, algunas de ellas de naturaleza analítica como lo es el caso del Método de los Momentos, y otras numéricas como es el Método de simulación de Montecarlo.

La propagación de la incertidumbre asociada a cada variable en el modelo matemático contempla los siguientes pasos:

Paso 1. Generar aleatoriamente un valor de cada variable desde sus respectivas distribuciones de probabilidad, para esto es necesario generar números aleatorios de probabilidad (entre 0 y 1) que se introducirán en la ecuación de la distribución acumulada inversa, que caracteriza a la variable en estudio.

Paso 2. Sustituir en la función g (X1, X2, X3,…, Xn), el conjunto de valores generados aleatoriamente y obtener un probable valor de Y.

Paso 3. Registrar y almacenar el valor resultante de Y.

Paso 4. Retornar al paso 1 y repetir nuevamente los tres primeros pasos hasta completar un número "m" de iteraciones.

Paso 5. Una vez completadas las "m" iteraciones, construir un histograma de frecuencias con los "m" probables valores de "Y" almacenados.

Paso 6. Aplicar el algoritmo de caracterización probabilística de variables aleatorias a la variable de salida.

### 1.8. Método de Simulación Montecarlo.

La simulación de Montecarlo (1) es una técnica que combina conceptos estadísticos (muestreo aleatorio) con la capacidad que tienen los ordenadores para generar números pseudo-aleatorios y automatizar cálculos. El nombre de Montecarlo proviene de la famosa ciudad de Mónaco, donde abundan los casinos de juego y donde el azar, la probabilidad y el comportamiento aleatorio conforman todo un estilo de vida. Los orígenes de esta técnica están ligados al trabajo desarrollado por el matemático Stanislaw Marcin Ulam[1] y John Von Neumann[2] a finales de los 40, cuando investigaban el movimiento aleatorio de los neutrones. En años posteriores, la simulación de Montecarlo se ha venido aplicando a una infinidad de ámbitos como alternativa a los modelos matemáticos exactos o incluso como único medio de estimar soluciones para problemas complejos. Así, en la actualidad es posible encontrar modelos que hacen uso

[1] (13 de abril de 1909 – 13 de mayo de 1984) matemático polaco–estadounidense.

[2] (28 de diciembre de 1903 - 8 de febrero de 1957) matemático húngaro-estadounidense.

de esta simulación en el área informática, empresarial, económica, industrial e incluso social. En otras palabras, este algoritmo está presente en todos aquellos ámbitos en los que el comportamiento aleatorio o probabilístico desempeña un papel fundamental.

La clave de la simulación de Montecarlo consiste en crear un modelo matemático del sistema, proceso o actividad que se desee analizar, identificando aquellas variables de entrada del modelo cuyo comportamiento aleatorio determina el comportamiento global del sistema. Una vez identificadas dichas variables aleatorias, se lleva a cabo un experimento consistente. Primero se genera con ayuda del ordenador muestras aleatorias (valores concretos) para dichas variables de entrada, analizando el comportamiento del sistema ante los valores generados. Tras repetir n veces este experimento, se dispondrán de n observaciones sobre el comportamiento del sistema, lo cual será de utilidad para entender el funcionamiento del mismo. Mientras mayor sea el número experimentos que se lleven a cabo, mucho más preciso será el análisis.

## 1.9. Herramientas y tecnologías a utilizar.

Durante el desarrollo de cualquier aplicación informática se hace necesaria la utilización de diversas herramientas y tecnologías que faciliten este proceso. Seguidamente se tratarán las utilizadas para el desarrollo de la solución propuesta, las cuales fueron previamente seleccionadas por un equipo de analistas y arquitectos del proyecto SIC y se contemplan en el documento: “Selección tecnológica del proyecto SIC”.

### 1.9.1. Lenguaje de programación. Java.

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. Provee una sintaxis similar a la de los lenguajes de programación C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel que suelen inducir a muchos errores como la manipulación directa de punteros a memoria (4). Es uno de los lenguajes de programación de mayor crecimiento contando con un amplio soporte, abundante documentación y un gran número de bibliotecas y herramientas para extender sus funcionalidades. Se ha mostrado ideal para desarrollar aplicaciones distribuidas basadas en red en un amplio rango de entornos desde los dispositivos de red

embebidos hasta los sistemas de sobremesa e Internet. El lenguaje cuenta con amplia documentación en línea y una activa comunidad.

### 1.9.2. Lenguaje de programación. Groovy.

Groovy es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Nace en 2003 de la mano de James Strachan y Bob McWhirter, y se estandariza en el JSR-241 un año después (5). Es un lenguaje dinámico que permite acceder al API (Interfaz para Programas de Aplicación, del inglés Application Programming Interface) estándar Java desde un enfoque diferente gracias al soporte de funcionalidades como la sobrecarga de operadores y la creación de tipos en tiempo de ejecución. Cualquier cosa que pueda hacerse con Groovy es posible también con Java, y viceversa. La diferencia entre ambos lenguajes está en su propia naturaleza, que provoca que el código necesario para una determinada tarea en Groovy sea en promedio unas cinco veces más breve que el equivalente en Java. (6)

Groovy es totalmente compatible con el código Java, pero añade características dinámicas y sintácticas presentes en otros lenguajes como Python, Smalltalk o Ruby, aportando una sintaxis que aumenta enormemente la productividad y un entorno de ejecución que permite manejar los objetos de formas que en Java serían extremadamente complicados.

Groovy cuenta con una creciente comunidad de desarrolladores y abundante documentación. Actualmente se encuentra liberada la versión 1.8.6 del 9 de Febrero de 2012 bajo la licencia Apache License v2.0.

### 1.9.3. Framework de desarrollo. GWT.

Google Web Toolkit (GWT) es un *framework* de desarrollo en Java de código abierto creado por Google en mayo de 2006 (7). Posibilita el desarrollo y depuración en cualquier entorno de desarrollo integrado de aplicaciones AJAX (Asynchronous JavaScript And XML) utilizando el lenguaje de programación Java. Entre sus principales características se encuentra la existencia de un compilador y traductor que convierte todo el código Java a JavaScript y HTML, el cual será compatible con cualquier navegador web, lo cual es notorio ya que cada navegador suele necesitar código específico para lograr un front-end correcto en una aplicación web. Además, cuenta con una amplia comunidad, abundante documentación en línea y una

gran cantidad de bibliotecas desarrolladas por Google o terceros para aumentar sus funcionalidades. Actualmente se encuentra liberada la versión 2.4.0 del 8 de Septiembre de 2011, lanzado bajo la licencia Apache License v2.0.

### 1.9.4. Framework de desarrollo. Grails.

Grails (8) es un *framework* para el desarrollo de aplicaciones web sobre la plataforma Java Enterprise Edition. Se encuentra desarrollado sobre el lenguaje de programación Groovy y se basa en el principio de reutilización de código para aumentar la productividad y disminuir los riesgos de desarrollo. Cuenta con una amplia comunidad que brindan soporte, actualmente se encuentra liberada la versión 2.0.3, lanzada bajo la licencia Apache License 2.0. Permite al programador abstraerse de gran parte de la configuración típica que incluyen los *frameworks* Modelo Vista Controlador (MVC). Además, se aprovecha de un lenguaje dinámico como Groovy para acortar los tiempos de desarrollo y simplemente dejarlos en escribir código, actualizar, testear y depurar fallos. Esto hace que el desarrollo de la aplicación sea mucho más ágil.

Para facilitar el trabajo Grails dispone de una arquitectura de plug-ins con una comunidad de usuarios que ofrecen plug-ins para seguridad, AJAX, testeo, búsqueda, informes y servicios web. Posibilitando añadir complicadas funcionalidades de manera sencilla. Entre sus plug-ins se encuentra gwt-grails que permite la integración de los *frameworks* de desarrollo Grails y GWT, con el fin de utilizar las potencialidades que ambos brindan.

### 1.9.5. Biblioteca. SSJ.

SSJ (Stochastic Simulation in Java) es una biblioteca de simulación estocástica, desarrollado en Java y FOLTRAN por la Universidad de Montreal (9). Está publicada bajo la licencia GNU GPL v2 y posee amplio y continuo soporte; actualmente se encuentra liberada la versión 2.5 del 16 de febrero de 2012. La biblioteca ofrece facilidades para la generación de variables aleatorias uniformes y no uniformes, soporte a diversas distribuciones de probabilidad, algoritmos de pruebas de bondad de ajuste y funciones matemáticas. La SSJ posee los principales algoritmos y funciones matemáticas de mayor procesamiento escritos en lenguaje FOLTRAN con el objetivo de reducir los tiempos de ejecución de sus funcionalidades.

### 1.9.6. IDE de desarrollo. Eclipse.

Eclipse es un Entorno de Desarrollo Integrado (IDE del inglés, Integrated Development Environment) de código abierto y multiplataforma, desarrollado originalmente por IBM y actualmente propiedad de la Fundación Eclipse (10). Cuenta con una arquitectura abierta y basada en módulos (plug-ins) posibilitando extender sus funcionalidades como son la incorporación de varios lenguajes de programación como Java, Groovy, C/C++ y Python; y diversos *frameworks* como GWT, Grails y QT.

Eclipse dispone de un editor de texto con resaltado de sintaxis, auto completamiento de código para varios lenguajes de programación, un compilador en tiempo real. Además de las extensiones para la realización de pruebas unitarias mediante JUnit y control de versiones con SVN (Subversion).

### 1.9.7. Lenguaje unificado de modelado. UML.

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés Unified Modeling Language) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software (11). Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. UML es un lenguaje más expresivo, claro y uniforme que los anteriores definidos para el diseño Orientado a Objetos, que no garantiza el éxito de los proyectos pero sí mejora sustancialmente el desarrollo de los mismos, al permitir una fuerte integración entre las herramientas, los procesos y los dominios.

### 1.9.8. Herramienta CASE. Visual Paradigm.

Visual Paradigm para UML (VP-UML) (12) es una herramienta CASE (Ingeniería de Software Asistida por Computación del inglés Computer Aided Software Engineering) profesional, fácil de utilizar y que soporta el ciclo de vida completo del desarrollo de software; usa UML como lenguaje de modelado lo cual ayuda a la construcción de aplicaciones con calidad, de manera intuitiva y a menor coste; además de que facilita la comunicación entre los miembros del equipo de desarrollo al garantizar el uso de un lenguaje estándar común. Cuenta con un diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad. Se integra fácilmente con distintos IDE de desarrollo como Eclipse. Posibilita la generación

de código para varios lenguajes de programación como Java y C#. Se encuentra disponible en múltiples plataformas como Microsoft Windows y Linux.

### 1.10. Metodología a Utilizar. RUP.

El Proceso Unificado de Desarrollo (RUP) es una metodología de desarrollo de software que está basado en componentes e interfaces bien definidas, y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos en la actualidad (13). RUP es el resultado de la experiencia de más de 30 años de trabajo de James Rumbaugh, Grady Booch e Ivar Jacobson. Se divide en cuatro fases (Inicio, Elaboración, Construcción, Transición), presenta nueve flujos de trabajos, de ellos, seis son de ingeniería (Modelamiento del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba, Instalación o Distribución) y tres de apoyo (Configuración y Administración de Cambio, Administración de Proyecto, Ambiente) y posee como características fundamentales que es centrado en la arquitectura, guiado por casos de uso e iterativo e incremental.

### 1.11. Conclusiones.

Con el análisis del estado del arte asociado a los principales conceptos de caracterización probabilística de variables aleatorias, propagación de la incertidumbre asociada a cada variable de los modelos matemáticos y la simulación de Montecarlo, se propone el desarrollo de un módulo de simulación de Montecarlo de código abierto, multiplataforma e integrada al proyecto SIC bajo los principios de soberanía tecnológica como solución al problema a resolver planteado. Para la realización de la misma se utilizará el lenguaje de programación Java complementado por la biblioteca SSJ para el desarrollo de los principales algoritmos relacionados con la simulación de Montecarlo, con la finalidad de lograr la mayor eficiencia y rendimiento en los cálculos matemáticos. El lenguaje Java integrado al framework GWT para el diseño de la interfaz de usuario y la visualización de los datos, con el objetivo de agilizar el desarrollo e integración de la misma; y el Lenguaje Groovy integrado al framework Grails para el desarrollo de la lógica del negocio. La metodología a utilizar para guiar el proceso de desarrollo es RUP, empleando como lenguaje de modelado UML y como herramienta CASE para visualizar los modelos, VP-UML.

# CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA.

## 2.1. Introducción.

En el presente capítulo se analizan los requisitos funcionales y no funcionales, se describe los algoritmos matemáticos: caracterización probabilística de variables aleatorias y la propagación de la incertidumbre asociada a cada variable en el modelo matemático, se describen los casos de usos y se detalla la arquitectura y los patrones de diseño utilizados. Se realiza la descripción del modelo de diseño de la solución propuesta.

## 2.2. Requisitos funcionales.

- Caracterizar probabilísticamente variables aleatorias.

La caracterización probabilística de variables aleatorias permite determinar la distribución de probabilidad que mejor representa a una variable aleatoria específica, permitiendo cuantificar además el nivel de incertidumbre asociado.

- Propagar la incertidumbre asociada a cada variable en un modelo matemático.

La propagación de la incertidumbre asociada a cada variable en un modelo matemático permite determinar la distribución de probabilidad que más se ajusta a un modelo matemático.

## 2.3. Especificación de los requisitos no funcionales.

Los requisitos no funcionales definen las cualidades que debería tener un producto. Son los llamados “cualidades de un sistema” (14). Estos son aspectos importantes que este debe cumplir un determinado producto para lograr un aprovechamiento óptimo de las funcionalidades del sistema teniendo en cuenta el entorno en el que será utilizado.

A continuación se enuncian, separados en categorías, todos los requisitos no funcionales que debe cumplir el sistema para su correcto funcionamiento:

**Usabilidad**:

La interfaz de usuario debe:

**RNF1.** Ser de fácil operación y basada en áreas de trabajo.

**RNF2.** Tener capacidad de escalabilidad a la resolución de la pantalla que se disponga.

**RNF3.** Poseer ayuda, incluyendo documentación, manuales de ingeniería de confiabilidad, según estándares que se apliquen.

**RNF4.** Tener teclas funcionales para el trabajo con la herramienta.

**RNF5.** Debe mostrar la explicación paso a paso de las operaciones que debe realizar el usuario correspondiente a las diferentes metodologías.

**Confiabilidad:**

**RNF6.** Tiempo de recuperación ante fallas: El sistema debe restablecer sus funciones 30 segundos después de haber ocurrido la falla. No se desea perder la operatividad por períodos prolongados. El tiempo máximo de espera de los usuarios para recuperación debe ser de 5 minutos.

**RNF7.** Disponibilidad operativa del sistema: El sistema debe estar disponible todos los días del año, las 24 horas del día.

**Desempeño:**

**RNF8.** Garantizar procesos y transacción masiva de datos: los procesos y transacción masiva de datos deben estar definidos hacia y desde todas las estaciones de trabajo y sistemas asociados, sin efectos negativos sobre el rendimiento del sistema.

**RNF9.** Tiempos de cálculos operacionales: Las respuestas de los cálculos de las diferentes aplicaciones deben estar dadas en menos de 3 segundos. El tiempo máximo de respuesta esperado por los usuarios es de 30 segundos.

**Soportabilidad:**

**RNF10.** El sistema debe contar con una arquitectura abierta y distribuida, modular, de capacidad escalable y tecnología actualizable de acuerdo con las necesidades operacionales y tendencias tecnológicas de las aplicaciones y componentes que se ejecutan o interactúan con el sistema.

**RNF11.** La programación del sistema debe estar orientada a objetos.

**RNF12.** Manejo integrado de datos: El sistema debe poseer funcionalidad de manejo integrado de datos complejos.

**Restricciones de diseño (Implementación):**

**RNF13.** Empleo del conjunto de herramientas y tecnologías definidas por el proyecto SIC: Se debe emplear el conjunto de herramientas y tecnologías definidas por el proyecto SIC, así como para futuros desarrollos. De manera que garantice la escalabilidad y compatibilidad entre los diferentes módulos.

**RNF14.** Rendimiento y alta disponibilidad: El sistema debe poseer un alto rendimiento y disponibilidad de manera que se garantice su operatividad de forma segura, efectiva y confiable.

**RNF15.** El sistema debe ser multiplataforma: El sistema debe ejecutarse en diversas plataformas de *hardware* y software, es decir, debe ser portable a varios sistemas operativos como Windows, Linux, Mac OS, Android.

**RNF16.** Licencia bajo código abierto: El sistema debe cumplir con los lineamientos necesarios para la producción de software libre y la comunidad de desarrollo y soporte, manteniendo la confidencialidad del negocio y las operaciones.

**Software:**

**RNF17.** El sistema requiere dos escenarios de despliegue.

Estos escenarios de despliegue serán las estaciones de trabajo y el trabajo con servidores. Los sistemas operativos que funcionarán en dichas estaciones de trabajo son los siguientes:

Para estaciones de trabajo: Nova, Ubuntu, Debian, y Windows.

Para servidores: Nova, Ubuntu, Debian, y Windows.

**RNF18.** Requiere que se emplee un navegador web, entre los soportados se encuentran Internet Explorer, Mozilla Firefox, Safari y Opera.

**Licencia:**

**RNF19.** El sistema debe cumplir con los lineamientos necesarios para la producción de software libre y la comunidad de desarrollo y soporte, manteniendo la confidencialidad y las operaciones de la institución, ya que es la política del centro.

**Requisitos Legales, de Derecho de Autor y otros:**

**RNF20.** Los requisitos legales, de derecho de autor y otros se establecen a través del Centro de Informática Industrial (CEDIN) y el Centro de Estudios de Ingeniería de Mantenimiento (CEIM).

**Estándares Aplicables:**

**RNF21.** Los estándares de calidad utilizados durante el desarrollo de la aplicación serán los establecidos por el Centro Nacional de Calidad de Software (Calisoft). Los mismos serán conciliados entre las partes durante la preparación de las pruebas.

## 2.4. Descripción de los Actores del Sistema.

| Actor | Descripción |
|---|---|
| Ingeniero de Confiabilidad. | Es la persona que posee la autoridad de realizar la caracterización probabilística de las variables aleatorias y la propagación de la incertidumbre en el modelo matemático. |

Tabla 1. Descripción de los actores del sistema.

## 2.5. Descripción de los Casos de Uso del Sistema.

El sistema cuenta con un actor y dos casos de uso como puede apreciarse en la Ilustración 8. El Ingeniero de Confiabilidad es el actor del sistema y este inicializa el caso de uso “Caracterizar variables aleatorias” y el caso de uso “Propagar incertidumbre” el cual a su vez en caso de ser necesario hace uso del caso de uso “Caracterizar variables aleatorias”.

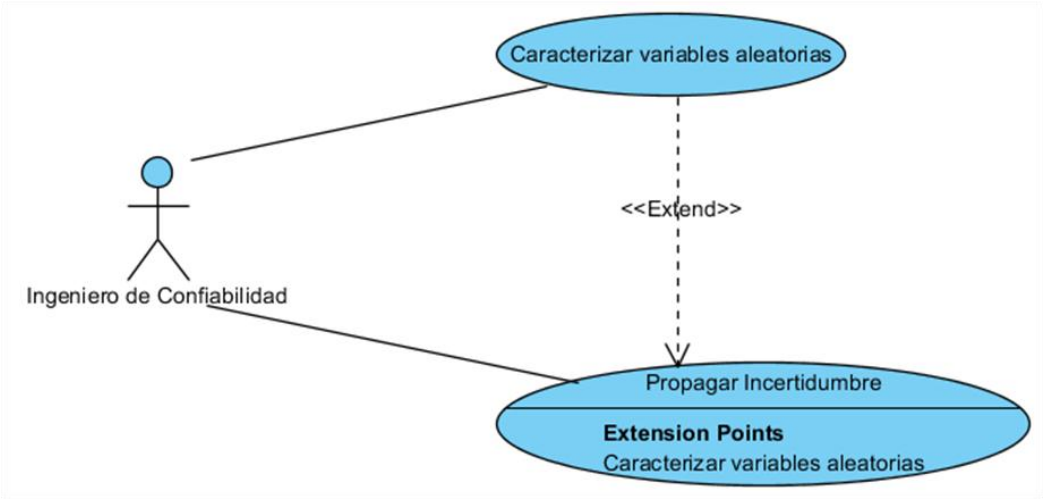


Ilustración 8. Diagrama de casos de uso.

### 2.5.1. Descripción del caso de uso: Caracterizar variables aleatorias.

| | |
|---|---|
| Caso de uso | Caracterizar variables aleatorias. |
| Actores | Ingeniero de Confiabilidad. |
| Descripción | Este caso de uso permite determinar la función de probabilidad que mejor representa a una variable aleatoria específica, permitiendo cuantificar además el nivel de incertidumbre asociado.<br><br>El caso de uso inicia cuando el actor desde su interfaz principal selecciona en el menú la opción "Herramientas", luego "Montecarlo" y selecciona la opción, "Caracterizar variables aleatorias". El sistema despliega la pantalla principal para la aplicación de la caracterización probabilística de variables aleatorias. |

| | |
|---|---|
| Complejidad | Alta |
| Prioridad | Crítico |
| Precondiciones | El actor se encuentra autenticado en el sistema. |
| Postcondiciones | |
| Flujo de eventos | |
| Flujo básico. | |

| Actor | Sistema |
|---|---|
| 1. Selecciona en el menú principal Herramientas->Montecarlo ->Caracterización de variables aleatorias. | 1.1. Muestra la interfaz de selección del tipo de caracterización a realizar: “Ajustar distribución a datos” o “Definir distribución”. |
| 2. Selecciona la opción “Ajustar distribución a datos”. En caso de seleccionar “Definir distribución” se pasaría a la sección 2. | 2.1. Muestra la interfaz de selección del tipo de variable aleatoria a caracterizar: continua o discreta. (por defecto se selecciona continua). |
| 3. Selecciona la opción “continua”. En caso de seleccionar “discreta” se pasaría a la sección 3. | 3.1. Muestra la interfaz de selección de todas las distribuciones de probabilidad continuas.<br>3.2. Muestra la interfaz de selección de las pruebas de bondad de ajuste.<br>3.3. Muestra la interfaz para la entrada de los datos históricos. |

| | |
|---|---|
| 4. Selecciona las distribuciones de probabilidad.<br>5. Selecciona las pruebas de bondad de ajuste a realizar.<br>6. Introduce el conjunto de datos históricos. | |
| 7. Presiona el botón “Caracterizar”. | 7.1. Valida que existan distribuciones de probabilidad seleccionadas. En caso de no existir distribuciones seleccionadas se pasaría al flujo alterno 1.<br>7.2. Valida que existan pruebas de bondad de ajuste seleccionadas. En caso de no existir pruebas seleccionadas se pasaría al flujo alterno 2.<br>7.3. Valida que se introdujeron al menos 15 datos históricos de la variable a caracterizar. En caso de ser menos se pasaría al flujo alterno 3.<br>7.4. Muestra una ventana emergente con el resultado de la caracterización. |
| 8. Presiona el botón “Cancelar”. | 8.1. Cierra la ventana.<br>8.2. Termina el caso de uso. |
| Sección 2: Definir distribución. | |
| 1. Selecciona la opción “Definir distribución” en el paso 2 del flujo básico. | 1.1. Muestra una pestaña con todas las distribuciones probabilísticas para variables continuas y otra para las |

| | |
|---|---|
| | variables discretas disponibles. |
| 2. Selecciona una distribución de probabilidad. | |
| 3. Presiona el botón “Definir”. En caso de presionar el botón “Cancelar” se pasaría al paso 8.1 del flujo básico. | 3.1. Muestra una interfaz solicitando los parámetros de la distribución de probabilidad. |
| 4. Ingresa los parámetros asociados a la distribución de probabilidad. | |
| 5. Presiona el botón “Caracterizar”. En caso de presionar el botón “Cancelar” se pasaría al paso 1 de la sección 2. | 5.1. Valida la integridad de los parámetros. En caso de existir errores se pasaría al flujo alterno 6.<br>5.2. Retorna al paso 7.4 del flujo básico. |
| Sección 3. Cambiar tipo de distribución. | |
| 1. Selecciona la opción “discreta” en el paso 3 del flujo básico. | 1.1. Muestra un mensaje informando que al cambiar la naturaleza de la variable se borrarán los datos históricos de la misma. |
| 2. Presiona el botón “Aceptar”. En caso de presionar el botón “Cancelar” retornaría al paso 3 del flujo básico. | 2.1. Muestra la interfaz de selección de todas las distribuciones de probabilidad discretas.<br>2.2. Retorna al paso 3.2 del flujo básico. |
| Flujos alternos | |
| No 1. No seleccionar distribución de probabilidad. | |
| 1. No selecciona ninguna distribución | 1.1. Muestra un mensaje informando |

| | |
|---|---|
| de probabilidad en el paso 4 del flujo básico. | que se debe seleccionar al menos una distribución de probabilidad para realizar la caracterización por ajuste. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 4 del flujo básico. |
| No 2. No seleccionar pruebas de bondad de ajuste. | |
| 1. No selecciona ninguna prueba de bondad de ajuste en el paso 5 del flujo básico. | 1.1. Muestra un mensaje informando que se debe seleccionar al menos una prueba de bondad de ajuste para realizar la caracterización por ajuste. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 5 del flujo básico. |
| No 3. Introducir menos de 15 datos. | |
| 1. Introduce menos de 15 datos en el paso 6 del flujo básico. | 1.1. Muestra un mensaje informando que se debe introducir 15 o más datos para realizar la caracterización por ajuste. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 6 del flujo básico. |
| No 4. Error en los parámetros. | |
| 1. Introduce parámetros con errores en el paso 6 de la sección 2. | 1.1. Muestra un mensaje informando que se debe introducir 15 o más datos para realizar la caracterización por ajuste. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 4 de la sección 2. |

| | | |
|---|---|---|
| Relaciones | CU Incluidos | No tiene |
| | CU Extendidos | No tiene |
| Requisitos funcionales | | |

Tabla 2: Descripción del caso de uso Caracterizar variables aleatorias.

### 2.5.2. Descripción del caso de uso: Propagar incertidumbre.

| | |
|---|---|
| Caso de uso | Propagar incertidumbre. |
| Actores | Ingeniero de Confiabilidad. |
| Descripción | Este caso de uso especifica la función que permite al actor aplicar el algoritmo de propagación de la incertidumbre a cada variable aleatoria en un modelo matemático.<br>El caso de uso inicia cuando el actor desde su interfaz principal selecciona en el menú la opción “Herramientas”, luego “Montecarlo” y selecciona la opción, “Propagar incertidumbre”. El sistema despliega la pantalla principal para la aplicación de la propagación de variables aleatorias. |
| Complejidad | Alta |
| Prioridad | Crítico |
| Precondiciones | El actor se encuentra autenticado en el sistema. |
| Postcondiciones | Si el caso de uso finaliza correctamente: Se aplicó la propagación de la incertidumbre a cada variable aleatoria en un modelo matemático.<br>Si el caso de uso no finaliza correctamente: Se mostró el mensaje de error correspondiente. |
| Flujo de eventos | |
| Flujo básico | |

| Actor | Sistema |
|---|---|
| 1. Selecciona en el menú principal Herramientas->Montecarlo->Propagar incertidumbre. | 1.1. Muestra la interfaz para introducir el modelo matemático (ecuación) a propagar. |
| 2. Introduce el modelo matemático a propagar. | |
| 3. Presiona el botón “Aceptar”. | 3.1. Valida el modelo matemático, en caso de presentar errores pasaría al flujo alterno 1.<br>3.2. Muestra una interfaz para introducir las variables del modelo y el número de iteraciones. |
| 4. Presiona el botón “Caracterizar Variable Aleatoria”. | 4.1. Retorna al paso 1 del flujo básico del caso de uso “Caracterizar variables aleatorias”.<br>4.2. Muestra los campos de las variables caracterizadas de color verde. |
| 5. Introduce el número de iteraciones. | |
| 6. Selecciona las pruebas de bondad de ajuste a realizar. | |

| | |
|---|---|
| 7. Presiona el botón “Simular”. | 7.1. Valida que cada variable se encuentre caracterizada. En caso de que alguna variable no se encuentre caracterizada se pasaría al flujo alterno 2.<br>7.2. Valida que existan un número de iteraciones válido. En caso de no existir un número de iteraciones válido se pasaría al flujo alterno 3.<br>7.3. Valida que existan pruebas de bondad de ajuste seleccionadas. En caso de no existir pruebas de bondad de ajuste seleccionadas se pasaría al flujo alterno 4.<br>7.4. Muestra una ventana emergente con el resultado de la simulación. |
| 8. Presiona el botón “Cerrar”. | 8.1. Cierra la ventana emergente.<br>8.2. Termina el caso de uso. |
| Flujos alternos | |
| No 1. Error en el modelo matemático. | |
| 1. Introduce el modelo matemático con errores en el paso 2 del flujo básico. | 1.1. Muestra un mensaje informando el error en el modelo matemático introducido. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 2 del flujo básico. |
| No 2. Error en las variables del modelo matemático. | |
| 1. No todas las variables se encuentran caracterizadas en el paso 4 del flujo | 1.1. Muestra un mensaje informando que todas las variables deben estar |

| | |
|---|---|
| básico. | caracterizadas. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 4 del flujo básico. |
| No 3. Error en número de iteraciones. | |
| 1. El usuario no introdujo un número de iteraciones válido en el paso 5 del flujo básico. | 1.1. Muestra un mensaje informando el error en el valor del número de iteraciones. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 5 del flujo básico. |
| No 4. No seleccionar pruebas de bondad de ajuste. | |
| 1. El usuario no seleccionó ninguna prueba de bondad de ajuste en el paso 6 del flujo básico. | 1.1. Muestra un mensaje informando que se debe seleccionar al menos una prueba de bondad de ajuste para realizar la caracterización por ajuste. |
| 2. Presiona el botón “Cerrar”. | 2.1. Retorna al paso 6 del flujo básico. |

| | | |
|---|---|---|
| Relaciones | CU Incluidos | No tiene |
| | CU Extendidos | Caracterizar variables aleatorias. |
| Requisitos funcionales | | |

Tabla 3: Descripción del caso de uso: Propagar incertidumbre.

## 2.6. Descripción de la arquitectura.

La solución se encuentra estructurada por dos subsistemas, la biblioteca de simulación para ingenieros en confiabilidad llamado Simulate (del español, Simular) y el subsistema de integración con la aplicación del proyecto SIC llamado Montecarlo.

El subsistema de la biblioteca Simulate se basa en una arquitectura orientada a componentes (15). La utilización de este estilo provee a la biblioteca de múltiples ventajas, en las que se destacan la reutilización

de código o componentes de terceros, fácil integración de componentes realizados por otros desarrolladores, aumenta la calidad de la solución, reduce el ciclo de desarrollo disminuyendo los costos del mismo. Además la facilidad de despliegue, ya que se puede sustituir cualquier componente por su nueva versión sin afectar al sistema o demás componentes.

El subsistema Montecarlo se basa en una arquitectura de N-Capas (16) y Cliente-Servidor (17). Este subsistema está estructurado por dos capas lógicas las cuales tiene una función específica y bien definida, estas capas son:

- **Capa de Presentación**: Es la encargada de generar la interfaz de usuario, en función de las acciones llevadas a cabo por el mismo.
- **Capa de Negocio**: Contiene todo la lógica del negocio y es donde se realiza todo el procesamiento necesario para atender las peticiones del usuario.

La capa de presentación se encuentra desacoplada de la lógica de negocio aumentado la reusabilidad de ambas en otros módulos compatibles, facilitando el despliegue y la actualización de los capas sin afectar a otras, facilita la extensión de las capas a partir de software de terceros, reduce el nivel de abstracción y la complejidad de las capas con el aislamiento de las mismas.

La arquitectura Cliente-Servidor brinda una mayor seguridad ya que los datos se almacenan en el servidor que ofrece más control sobre los mismos, cuenta con un acceso centralizado a los datos que están almacenados en el servidor lo que facilita su acceso y actualización, y facilita el mantenimiento ya que los roles y las responsabilidades se distribuyen entre los distintos servidores a través de la red lo que permite que un cliente no se vea afectado por un error en un servidor particular.

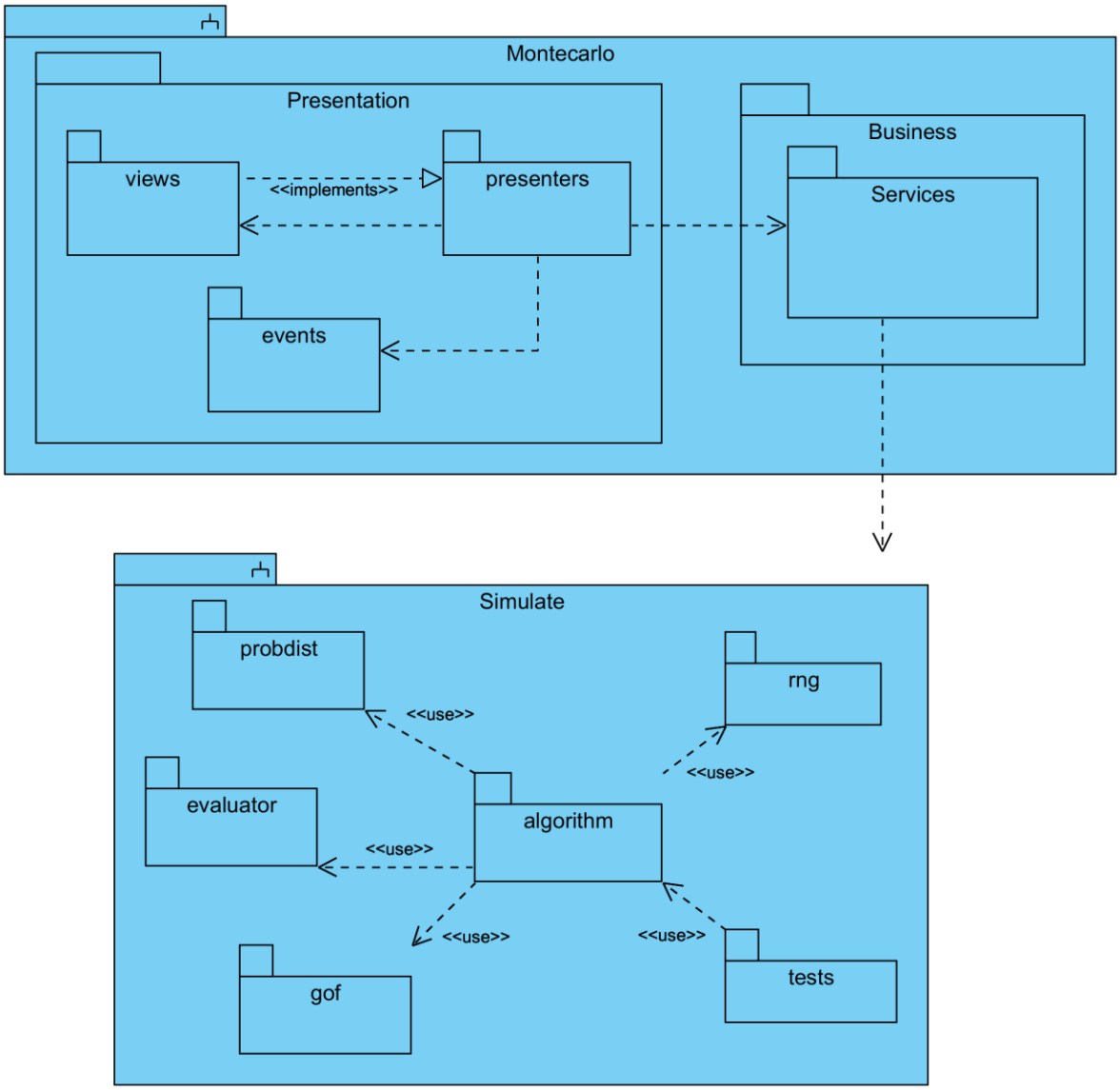


Ilustración 9. Diagrama de paquetes del sistema.

### 2.6.1. Capa de presentación.

La capa de presentación se encuentra estructurada en componentes que implementan las funcionalidades para que los usuarios interactúen con la aplicación; y dichos componentes están divididos en varias sub-capas aplicando el patrón Modelo Vista Presentador (MVP) (18).

Mediante el uso de este patrón se separa el manejo de eventos del sistema, la presentación y las acciones basadas en la interacción con el usuario en tres componentes separados. La Vista le delega a su Presentador toda la responsabilidad del manejo de los eventos del usuario. Mientras que el presentador es responsable de actualizar a la vista cuando ocurren cambios o se lanza algún evento del sistema.

Al implementar este patrón, se identifican los siguientes componentes:

- IView: Interfaz mediante la cual el presentador se comunica con la vista.
- View: Vista que implementa la interfaz IView y se encarga de manejar los aspectos visuales. Mantiene una referencia a su Presenter al cual le delega la responsabilidad del manejo de los eventos.
- Presenter: Contiene la lógica para responder a los eventos y manipula el estado de la vista mediante una referencia a la interfaz IView.

### 2.6.2. Capa de Negocio.

La capa de negocio tiene la responsabilidad de representar conceptos del negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio. Básicamente se puede ver como una capa intermedia que implementa las funcionalidades principales del sistema y encapsula toda la lógica del negocio relevante. En ella residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados.

## 2.7. Descripción del diseño.

En el presente epígrafe se describen los principales subsistemas y paquetes utilizados en la implementación del sistema, partiendo de un empaquetamiento de las clases del diseño para lograr un mejor entendimiento del código fuente de la solución, así como, las clases, atributos y funcionalidades más importantes.

Con el análisis de los requerimientos, casos de usos y el documento de la arquitectura, se conforma el empaquetamiento de la solución. La misma se conforma por dos subsistemas principales y sus respectivos paquetes que responden a las necesidades planteadas por el cliente, los mismos son:

Subsistema Simulate: biblioteca de simulación de Montecarlo para ingenieros en confiabilidad, la cual presenta funcionalidades como: algoritmo de caracterización probabilística de variables aleatorias y el algoritmo de propagación de la incertidumbre asociada a las variables de entrada de los modelos matemáticos. La biblioteca se encuentra estructurada por los siguientes paquetes:

- Paquete algorithm: contiene los algoritmos fundamentales que realiza la biblioteca.
- Paquete probdist: conjunto de distribuciones de probabilidad con funcionalidades específicas en el área de la confiabilidad integral.
- Paquete gof: conjunto de pruebas de bondad de ajuste.
- Paquete rng: conjunto de funciones para la generación de números aleatorios para las distribuciones de probabilidad.
- Paquete evaluator: conjunto de funciones para la evaluación de expresiones matemáticas.
- Paquete test: suite de pruebas de unidad a Simulate.

Subsistema Montecarlo: subsistema que contiene los componentes para la presentación de las interfaces visuales y la lógica de negocio.

- Paquete Presentation: paquete donde se manejan las acciones de los usuarios sobre la interfaz y visualiza los datos. Este engloba los siguientes paquetes:
  - views: paquete donde se encuentran las clases encargadas de mostrar al usuario los componentes visuales de la aplicación. Mantiene una referencia a su presentador, al cual le delega la responsabilidad del manejo de los eventos.
  - presenters: paquete que contiene las clases encargadas de manejar la lógica para responder a los eventos del usuario y manipula el estado de las vistas. Realiza peticiones a los servicios del servidor.
- Paquete Business: paquete donde se maneja toda la lógica de negocio en el proceso de simulación de Montecarlo. Este engloba los siguientes paquetes:

- services: contiene la lógica de negocio del sistema y responde a las peticiones de los presentadores, mediante el uso de la biblioteca de simulación de Montecarlo.

En la Ilustración 10 se aprecian con más claridad los subsistemas y paquetes antes mencionados, y las relaciones existentes entre ellos.

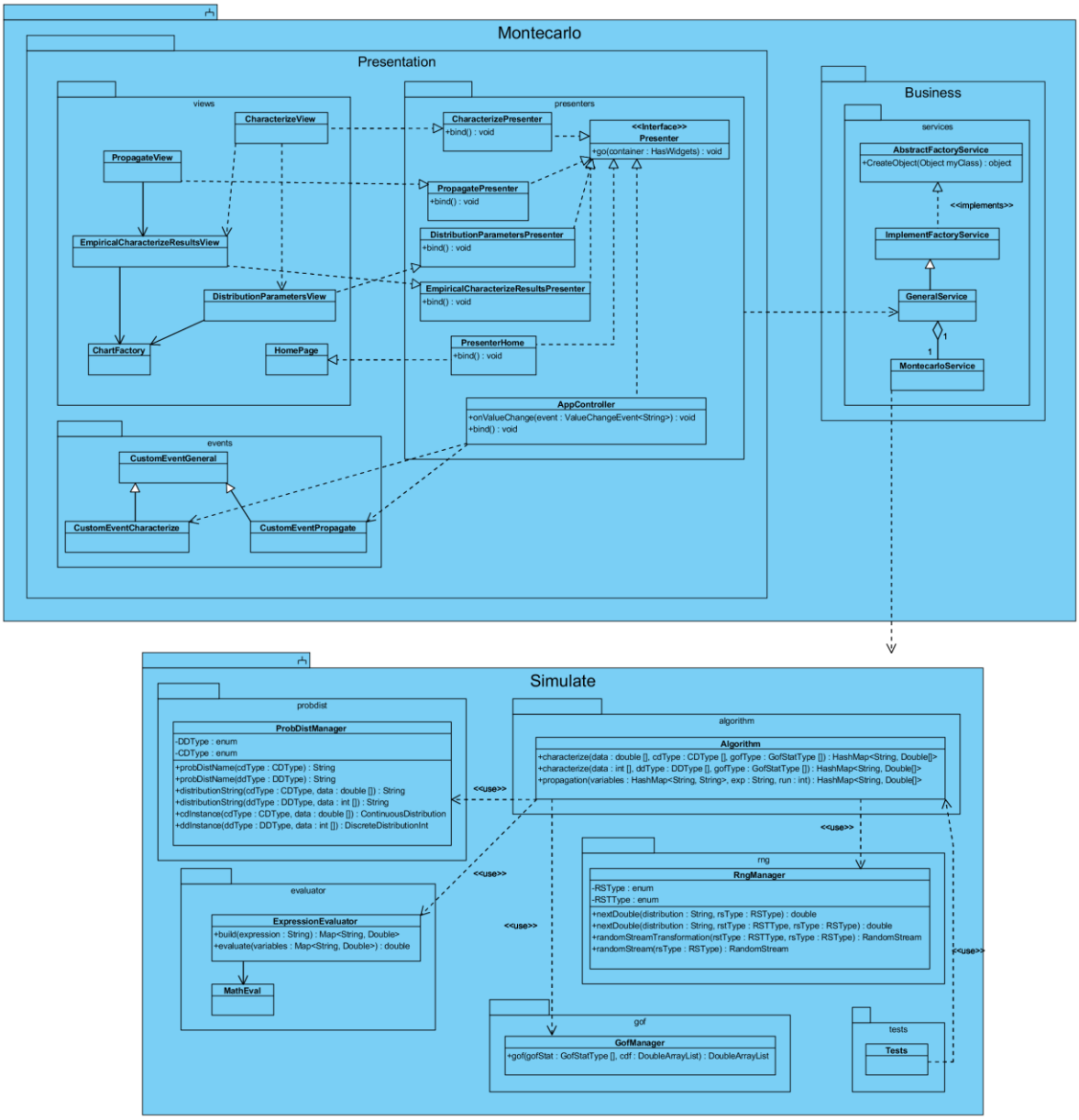


Ilustración 10. Diagrama de clases del diseño del Sistema.

## 2.8. Descripción de las clases del diseño.

En los siguientes epígrafes se describen las principales clases del diseño que responden a los Casos de Uso del Sistema.

### 2.8.1. Subsistema Simulate.

El subsistema Simulate (del español, Simular) constituye el núcleo de la herramienta matemática, está compuesto por los paquetes: algorithm, probdist, gof, rng, evaluator y tests; como se aprecia en la Ilustración 11.

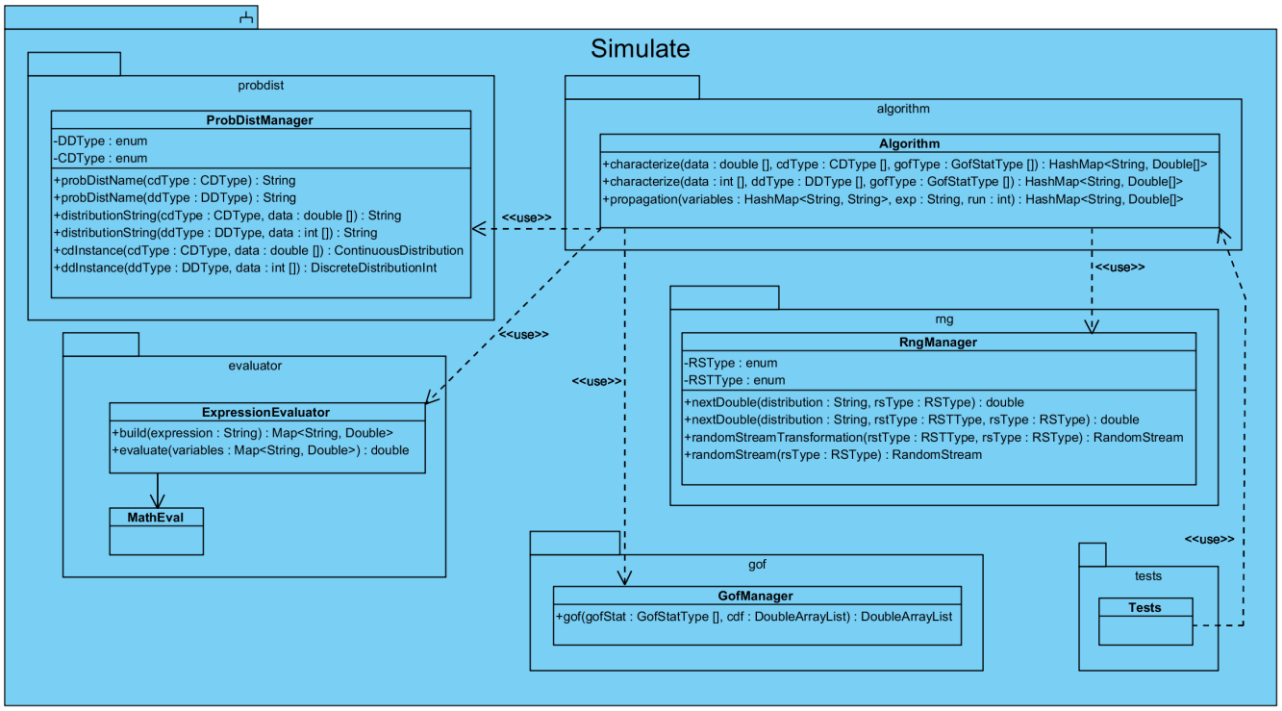


Ilustración 11. Diagrama de clases del diseño del subsistema Simulate.

#### 2.8.1.1. Paquete algorithm.

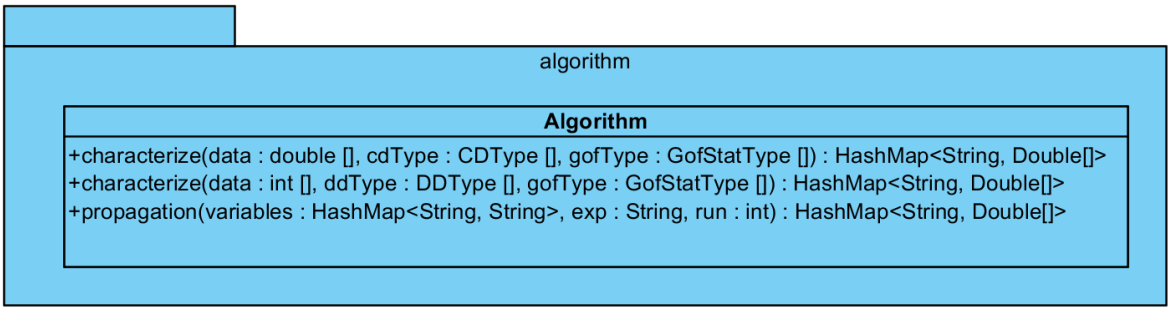


Ilustración 12. Diagrama de clases del diseño del paquete algorithm.

Este paquete contiene la clase Algorithm, la cual se encuentra descrita en la Tabla 4.

| Nombre: Algorithm | |
|---|---|
| Descripción: Clase que contiene los algoritmos que brinda la biblioteca. | |
| Tipo de clase: Controladora. | |
| Atributo | Tipo |
| | |
| Para cada responsabilidad | |
| Nombre: | characterize (data, cdType, gofType) |
| Descripción: | Función que retorna un mapa con los datos generados en el proceso de caracterización probabilística de la variable aleatoria de naturaleza continua dados los datos históricos de dicha variable, las distribuciones de probabilidad y las pruebas de bondad de ajuste a probar. |
| Nombre: | characterize (data, ddType, gofType) |
| Descripción: | Función que retorna un mapa con los datos generados en el proceso de caracterización probabilística de la variable aleatoria de naturaleza |

| | |
|---|---|
| | discreta dados los datos históricos de dicha variable, las distribuciones de probabilidad y las pruebas de bondad de ajuste a probar. |

Tabla 4. Descripción de la clase Algorithm.

#### 2.8.1.2. Paquete gof.

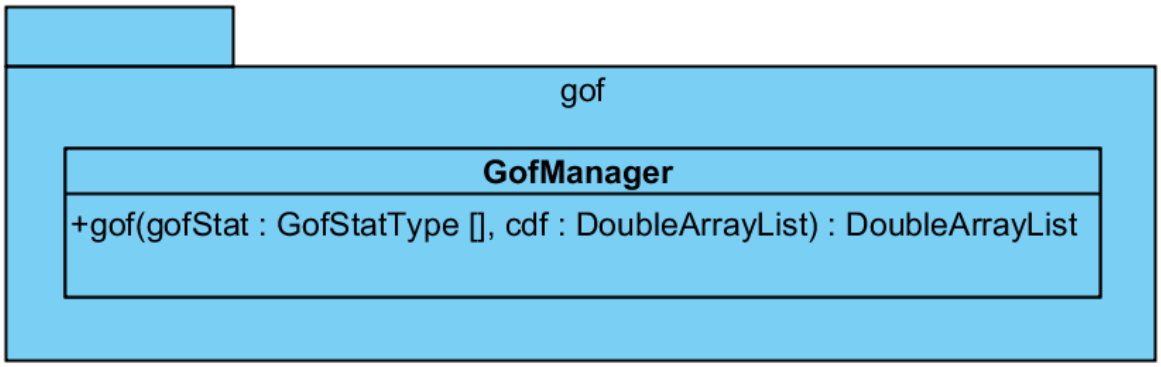


Ilustración 13. Diagrama de clases del diseño del paquete gof.

Contiene la clase GofManager, la cual se describe en la Tabla 5.

La clase GofManager brinda soporte a las pruebas de bondad de ajuste: Anderson Darling, Cramer Von Mises, Kolmogorov Smirnov, Watson G y Watson U.

| Nombre: GofManager | |
|---|---|
| Descripción: Clase para las pruebas de bondad de ajuste. | |
| Tipo de clase: Entidad. | |
| Atributo: | Tipo: |
| | |
| Para cada responsabilidad | |

| | |
|---|---|
| Nombre: | Gof (gofStat, cdf) |
| Descripción: | Función que retorna una lista con los datos generados por las pruebas de bondad de ajuste dada las pruebas de bondad de ajuste a realizar y la función de distribución. |

Tabla 5. Descripción de la clase GofManager.

#### 2.8.1.3. Paquete probdist.

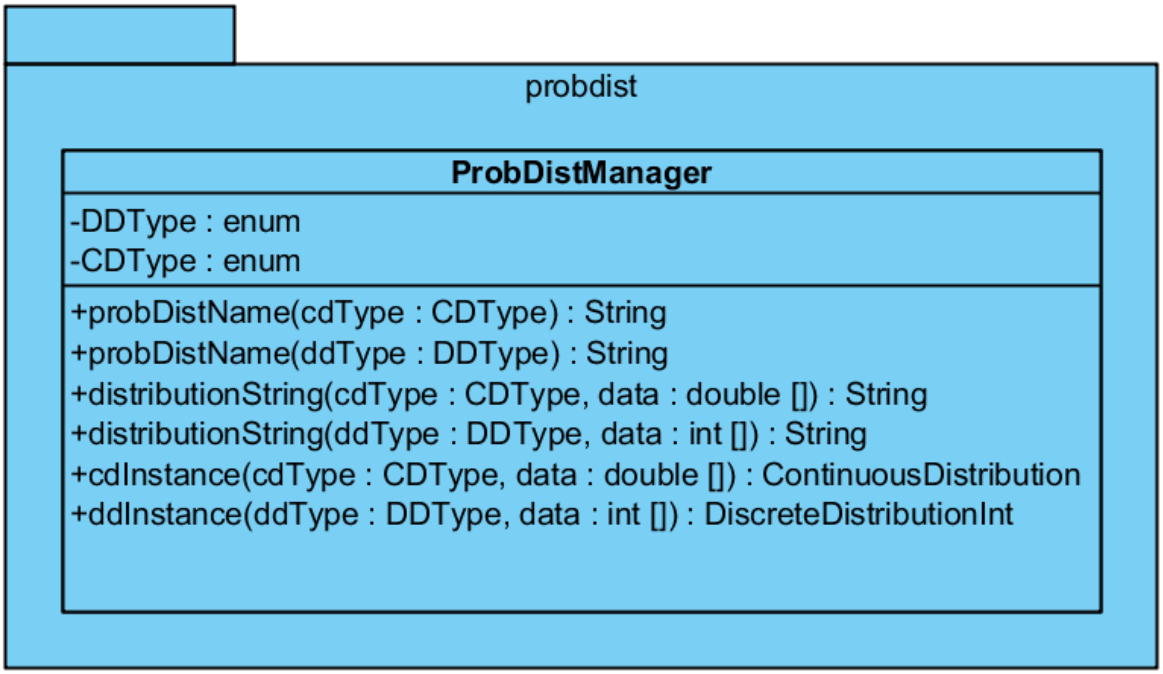


Ilustración 14. Diagrama de clases del diseño del paquete probdist.

Este paquete contiene la clase ProbDistManager, la cual se describe en la Tabla 6.

La clase ProbDistManager brinda soporte a las distribuciones de probabilidad:

De naturaleza discreta: Bernoulli, Binomial, Geometric, Logarithmic, Negativebinomial, Pascal, Poisson y Uniformint.

De naturaleza continua: Beta, Betasymmetrical, Cauchy, Chi, Chisquare, Erlang, Exponential, Extremevalue, Gamma, Gumbel, Hyperbolicsecant, Inversegamma, Inversegaussian, Laplace, Logistic, Loglogistic, Lognormal, Normal, Normalinversegaussian, Pareto, Pearson5, Pearson6, Student, Triangular, Uniform y Weibull.

| Nombre: ProbDistManager. | |
|---|---|
| Descripción: Clase para el manejo de las distribuciones de probabilidad. | |
| Tipo de clase: Entidad. | |
| Atributo: | Tipo: |
| DDType | enum |
| CDType | enum |
| Para cada responsabilidad | |
| Nombre: | probDistName (cdType) |
| Descripción: | Función que retorna el nombre de la distribución de probabilidad continua, dado un tipo de distribución de probabilidad continua pasada por parámetro. |
| Nombre: | probDistName (ddType) |
| Descripción: | Función que retorna el nombre de la distribución de probabilidad discreta, dado un tipo de distribución de probabilidad discreta pasada por parámetro. |
| Nombre: | distributionString (cdType, data) |
| Descripción: | Función que retorna el nombre de la distribución de probabilidad continua y sus atributos, dado un tipo de distribución de probabilidad continua. |

| | |
|---|---|
| Nombre: | distributionString (ddType, data) |
| Descripción: | Función que retorna el nombre de la distribución de probabilidad discreta y sus parámetros, dado un tipo de distribución de probabilidad discreta. |
| Nombre: | cdInstance (cdType, data) |
| Descripción: | Función que retorna una instancia de una distribución de probabilidad continua, dado el tipo de distribución y un set de datos flotantes. |
| Nombre: | ddInstance (ddType, data) |
| Descripción: | Función que retorna una instancia de una distribución de probabilidad discreta, dado el tipo de distribución y un set de datos enteros. |

Tabla 6. Descripción de la clase ProbDistManager.

#### 2.8.1.4. Paquete rng.

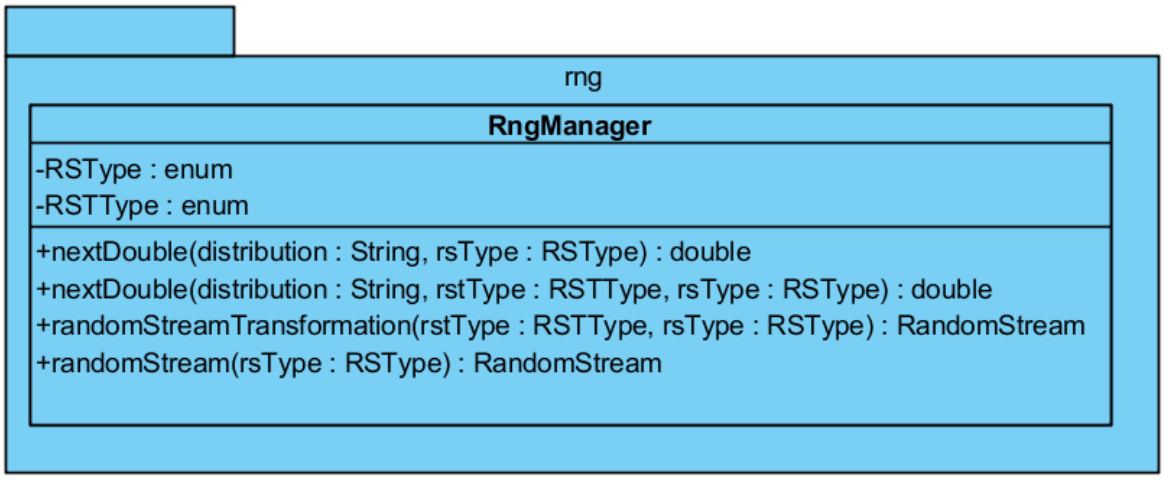


Ilustración 15. Diagrama de clases del diseño del paquete rng.

Este paquete contiene la clase RngManager, la cual se describe en la Tabla 7.

| Nombre: RngManager. | |
|---|---|
| Descripción: Clase para la generación de números aleatorios. | |
| Tipo de clase: Entidad. | |
| Atributo: | Tipo: |
| RSType | enum |
| RSTType | enum |
| Para cada responsabilidad | |
| Nombre: | randomStream (rsType) |
| Descripción: | Función que retorna una instancia de una función de generación de números aleatorios dado su tipo. |
| Nombre: | randomStreamTransformation (rstType, rsType) |
| Descripción: | Función que retorna una instancia de una función de generación de números aleatorios dado su tipo y la función de generación de números aleatorios. |
| Nombre: | nextDouble (distribution, rsType) |
| Descripción: | Función que retorna un valor aleatorio generado a partir de la distribución de probabilidad y el tipo de función de generación de números aleatorios dados. |
| Nombre: | nextDouble (distribution, rstType, rsType) |
| Descripción: | Función que retorna un valor aleatorio generado a partir de la distribución de probabilidad, el tipo de transformación y el tipo de función de generación |

| | |
|---|---|
| | de números aleatorios dados. |

Tabla 7. Descripción de la clase RngManager.

#### 2.8.1.5. Paquete evaluator.

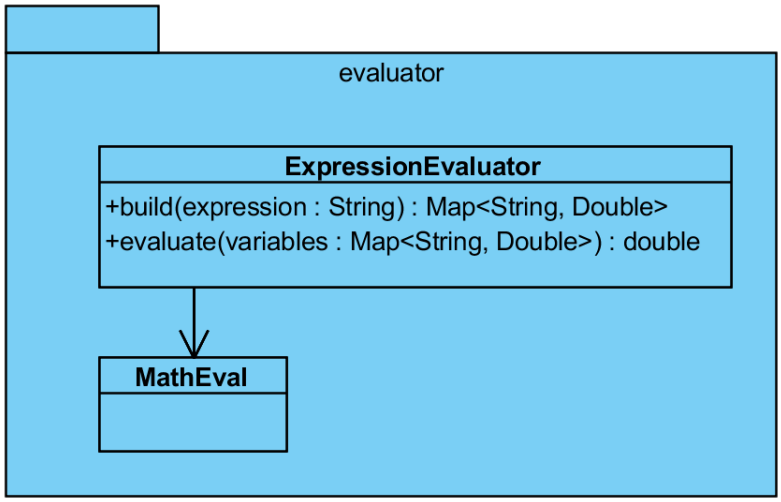


Ilustración 16. Diagrama de clases del diseño del paquete evaluator.

Este paquete contiene la clase ExpressionEvaluator, la cual se describe en la Tabla 8.

| Nombre: ExpressionEvaluator. | |
|---|---|
| Descripción: Clase para evaluar expresiones matemáticas. | |
| Tipo de clase: Entidad. | |
| Atributo: | Tipo: |
| | |
| Para cada responsabilidad | |
| Nombre: | build (expression) |
| Descripción: | Función que retorna un mapa con las variables que |

| | |
|---|---|
| | contiene una expresión dada y guarda dicha expresión. |
| Nombre: | evaluate (variables) |
| Descripción: | Función que retorna el resultado de la evaluación de una expresión matemática definida dado el valor de las variables que contiene. |

Tabla 8. Descripción de la clase ExpressionEvaluator.

### 2.8.2. Paquete Presentation.

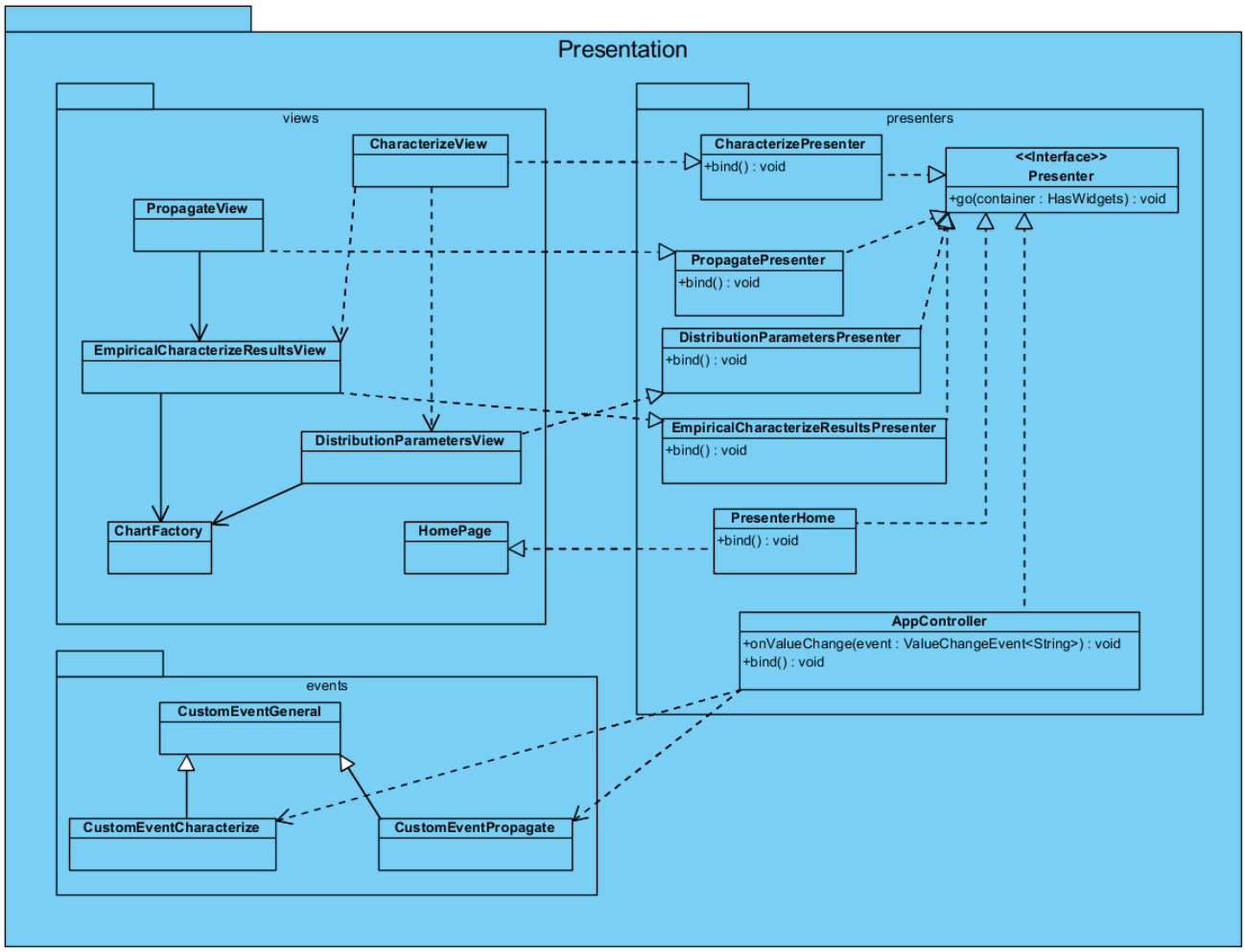


Ilustración 17. Diagrama de clases del diseño del subsistema Presentation.

### 2.8.2.1. Paquete views.

El paquete views contiene las clases que se encargan de mostrar al usuario las principales interfaces para caracterizar variables aleatorias y propagar la incertidumbre en los modelos matemáticos, como se aprecia en la Ilustración 17. Las clases principales que contiene este paquete son:

PropagateView: clase que implementa la interfaz del presentador PropagetePresenter y se encarga de mostrar al usuario los componentes visuales que responden al Caso de Uso: Propagar Incertidumbre.

CharacterizeView: clase que implementa la interfaz del presentador CharacterizePresenter y se encarga de mostrar al usuario los componentes visuales que responden al Caso de Uso: Caracterizar variables aleatorias. Además depende de las clases EmpiricalCharacterizeResultsView y DistributionParametersView.

#### 2.8.2.2. Paquete presenters.

El paquete presenters contiene las clases encargadas de manejar la lógica de las vistas, el lanzamiento de eventos y la interacción con los servicios del servidor. Este contiene la clase interfaz Presenter que es redefinida por el resto de las clases del paquete. Entre las clases principales que contiene este paquete se encuentran: AppController que se encarga de definir las acciones a desarrollar una vez lanzado un evento del sistema, la clase CharacterizePresenter que se encarga de responder a los eventos de la vista CharacterizeView y la clase PropagatePresenter encargada de responder a los eventos de la vista PropagateView.

#### 2.8.2.3. Paquete events.

El paquete events contiene las clases encargadas de manejar los eventos que lanzan los presentadores. Este contiene la clase interfaz CustomEventGeneral, del cual heredan las clases CustomEventCharacterize y CustomEventPropagate. CustomEventCharacterize se encarga de que cuando sea lanzada se muestre la vista CharacterizeView, mientras que CustomEventPropagate una vez lanzada se muestra la vista PropagateView.

### 2.8.3. Paquete Business.

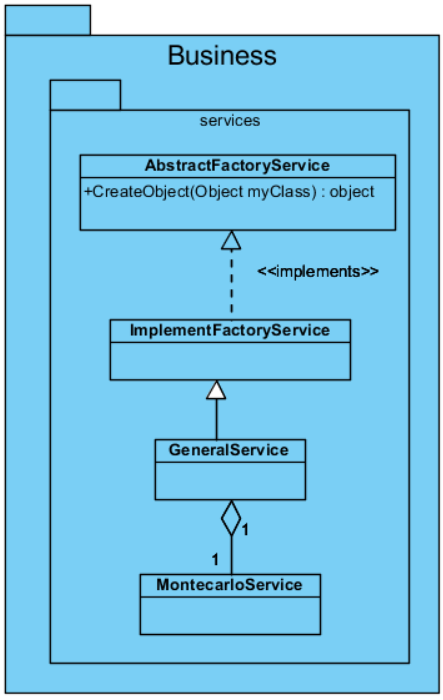


Ilustración 18. Diagrama de clases del diseño del subsistema Business.

#### 2.8.3.1. Paquete services.

El paquete services se encuentra compuesto por la clase GeneralService que hereda de la clase ImplementFactoryService y a su vez agrega la clase MontecarloService. La clase ImplementFactoryService implementa la clase interfaz AbstractFactoryService. Como se aprecia en la Ilustración 18.

## 2.9. Conclusiones.

En la etapa de análisis y diseño se realiza un análisis del sistema ya en términos de solución. En este capítulo quedó definida la estructura del sistema en el análisis, donde se proporciona una comprensión detallada de los requisitos con los cuales tiene que cumplir el sistema. Como resultado se obtuvieron varios artefactos: descripción de los requisitos funcionales y no funcionales, descripción de los actores del

sistema, diagrama y descripción de los casos de uso, descripción de la arquitectura, y el diagrama y descripción de las principales clases del diseño. El mayor aporte de este capítulo es que queda definido el análisis y diseño del sistema para su implementación.

# CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS.

## 3.1. Introducción.

En el presente capítulo se analiza el flujo de trabajo implementación, el cual denota el estado actual del sistema en términos de componentes y subsistemas de implementación. Además se definen las pruebas del software como elemento crítico para validación de la calidad del mismo, y revisión final del cumplimiento de las especificaciones del diseño y de la codificación.

## 3.2. Modelo de componentes.

La solución propuesta se encuentra estructurada por: la aplicación SIC, las bibliotecas Simulate y SSJ, el subsistema Montecarlo y los frameworks GWT y Grails. Como se aprecia en la Ilustración 19. La aplicación SIC hace uso de las funcionalidades caracterizar probabilísticamente variables aleatorias y propagar la incertidumbre asociada a cada variable del modelo matemático que provee el subsistema Montecarlo. Este se divide en los componentes Presentation y Business. Presentation hace uso de la framework visual GWT para la creación de las interfaces de usuario mientras que Business a su vez hace uso del framework Groovy y de la biblioteca Simulate para manejar toda la lógica de negocio del sistema.

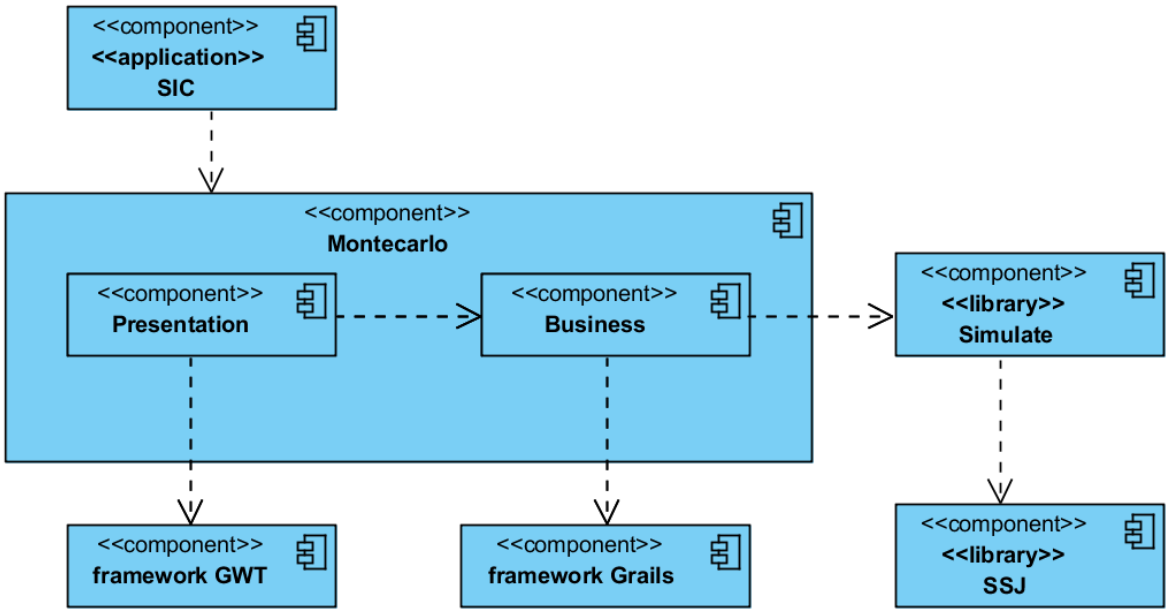


Ilustración 19. Diagrama de componentes de la herramienta de simulación de Montecarlo.

### 3.3. Estandarización y documentación.

Un estándar de codificación es una tecnología, formato o método desarrollado, adoptado a través de proceso abierto de consenso, con la ventaja de facilitar la legibilidad, mantenibilidad, interoperabilidad y distribución del código fuente de cualquier aplicación. Los estándares abiertos son capaces de soportar y administrar en forma más sencilla y menos costosa la creciente complejidad de los sistemas, constituyendo una especie de “garantía universal” de compatibilidad, ajena a cualquier proveedor de la industria por sí mismo.

La estandarización del código fuente repercute directamente en lo bien que un programador comprende un sistema de software, la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Por lo que, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y posteriormente se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Como elemento de apoyo a las técnicas de codificación en la facilitación de la inteligibilidad del código se encuentra la documentación interna de un software. Documentar el código de un programa es añadir suficiente información como para explicar lo que hace, de forma que no sólo los ordenadores sepan qué hacer, sino que además los humanos entiendan qué están haciendo y por qué. En el mundo del software todo programa que tenga éxito será modificado en el futuro, bien por el programador original o por otro programador que le sustituya, por lo que es importante que el programa se entienda para poder repararlo y modificarlo en el menor tiempo posible.

Con el objetivo de facilitar la comprensión, mantenimiento e inteligibilidad del código fuente de la herramienta de simulación de Montecarlo, se utiliza como estándar de codificación Java-Conventions (19), propuesto por Sun Microsystems. Para la automatización y evaluación del estándar de codificación durante el ciclo de vida del software se usa la funcionalidad para formatear el estilo de código que provee el IDE Eclipse.

### 3.4. Modelo de despliegue.

La aplicación se encontrará distribuida en dos nodos principales: el nodo PC Cliente encargado del funcionamiento del navegador Web donde será visualizada la aplicación y el nodo Servidor de Aplicaciones con la funcionalidad de soportar sobre él la aplicación Web, en este es donde se montará el sistema. En la Ilustración 20 se muestra la manera en que quedan distribuidos estos nodos y sus relaciones de comunicación.

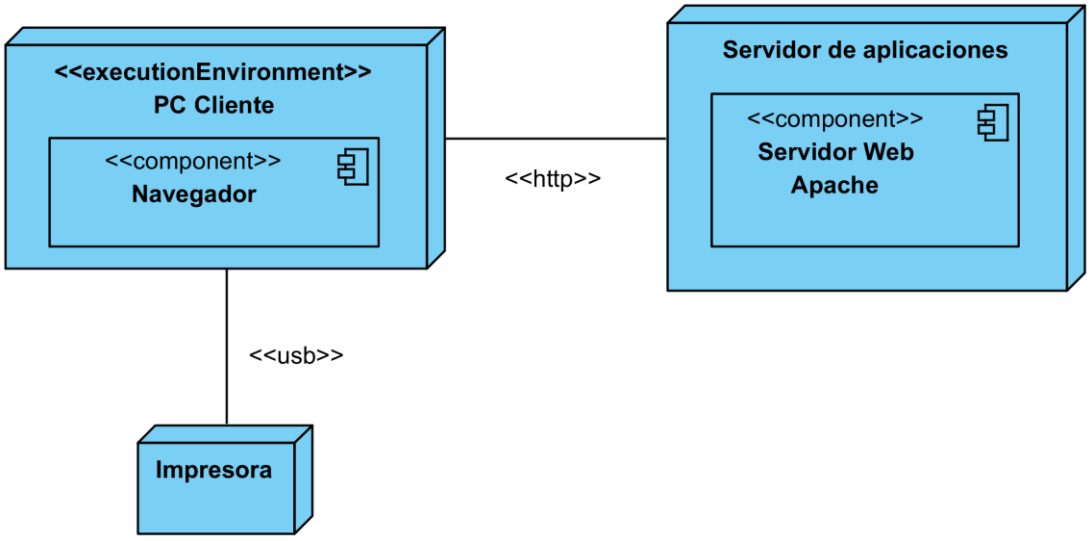


Ilustración 20. Diagrama de despliegue del módulo de simulación de Montecarlo.

### 3.5. Proceso de Pruebas.

Las pruebas de software consisten en una serie de actividades en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, realizando una evaluación del sistema o del componente probado. Su principal objetivo es evaluar o valorar la calidad del producto.

### 3.6. Métodos de pruebas.

Pruebas de **Caja Blanca**: Es un método de diseño de casos de prueba que se centran en los detalles procedimentales del software para obtener los casos de prueba, por lo que su diseño está fuertemente ligado al código fuente. El probador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. Al estar basadas en una implementación concreta, si ésta se modifica, por regla general las pruebas también deberán rediseñarse.

Pruebas de **Caja Negra**: Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Estas pruebas examinan algunos aspectos del modelo, fundamentalmente del sistema, sin tener mucho en cuenta la estructura interna del software.

Las pruebas de Caja Negra no constituyen una alternativa a las técnicas de prueba de Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de pruebas de Caja Blanca. Estas pruebas permiten encontrar funciones incorrectas o ausentes así como también errores de interfaz, errores en estructuras de datos o en accesos a las bases de datos externas. Además errores de rendimiento y errores de inicialización y terminación.

## 3.7. Pruebas de Unidad.

En el proceso de construcción y validación del software, la implementación y realización de pruebas unitarias es, entre otros, uno de los pilares fundamentales. La creación de pruebas unitarias y su ejecución automatizada permiten dotar a dicho proceso de confianza y calidad. Con estas pruebas no se encontrarán todos los errores que tenga el producto, pero ayudan a asegurar que cada una de las partes individuales que conforman el sistema funcionan correctamente.

### 3.7.1. Diseño de los Casos de Prueba.

Para la realización de las pruebas de unidad se utilizan como recursos físicos: un computador con microprocesador Intel Core 2 Duo de velocidad de la Unidad Central de Procesamiento (CPU del inglés, Central Processing Unit) de 2.20 GHz, Memoria de Acceso Aleatorio (RAM del inglés, Random Access Memory) de 1024 Mb y un disco duro con capacidad de 160 Gb y como recursos lógicos: sistema operativo Windows 7 Ultimate.

Se realizaron cuatro ciclos de pruebas, de ellos se automatizaron las pruebas de unidad, tomando como valores de comprobación los arrojados por herramientas de confiabilidad usadas en el mundo (Cristal Ball (20) y @Risk (21)).

| Aspecto/Función a probar: Funcionalidades de la biblioteca Simulate. | | | | |
|---|---|---|---|---|
| Casos de uso asociados:<br>Caracterizar variables aleatorias.<br>Propagar incertidumbre. | | | | |
| Caso # | Pasos, Procedimiento o Precondiciones de ejecución de la prueba | Datos de Entrada | Resultados Esperados | Resultados Obtenidos |
| 1 | Funcionalidad:<br>characterize(data, cdType, gofType) | data = [21.6, 63.0, 65.1, 83.3, 120.5, 121.0, 135.1, 184.2, 246.4, 298.5, 373.6, 430.6, 434.0, 446.8, 516.8, 597.9, 629.7, 647.6, 719.7, 737.6, 746.6, 756.7, 758.8, 977.9, 1082.0, 1082.0, 1178.0, 1282.0, 1373.0, 1447.0, 1519.0, 1589.0, 1676.0, 1769.0, 1789.0, 1832.0, 2072.0, 2259.0, 2290.0, 2554.0, 2773.0, 2894.0, 2939.0, 2969.0, 3438.0, 3595.0, 4083.0, 5804.0, 6415.0]<br>cdType = [BETA, BETASYMMETRICAL, CAUCHY, CHI, CHISQUARE, ERLANG, EXPONENTIAL, EXTREMEVALUE, GAMMA, GUMBEL, HYPERBOLICSECANT, INVERSEGAMMA, INVERSEGAUSSIAN, LAPLACE, LOGISTIC, LOGLOGISTIC, LOGNORMAL, NORMAL, NORMALINVERSEGAUSSIAN, PARETO, PEARSON5, PEARSON6, STUDENT, UNIFORM, WEIBULL] | LoglogisticDist<br>ErlangDist<br>GammaDist<br>GammaDist<br>ErlangDist | LoglogisticDist<br>ErlangDist<br>GammaDist<br>GammaDist<br>ErlangDist |

| | | | | |
|---|---|---|---|---|
| | | gofType = [AndersonDarling, CramerVonMises, KolmogorovSmirnov, WatsonG, WatsonU] | | |
| 2 | Funcionalidad:<br>characterize(data, ddType, gofType) | data = [21, 63, 65, 83, 120, 121, 135, 184, 246, 298, 373, 430, 434, 446, 516, 597, 629, 647, 719, 737, 746, 756, 758, 977, 1082, 1082, 1178, 1282, 1373, 1447, 1519, 1589, 1676, 1769, 1789, 1832, 2072, 2259, 2290, 2554, 2773, 2894, 2939, 2969, 3438, 3595, 4083, 5804, 6415]<br>ddType = [BERNOULLI, BINOMIAL, GEOMETRIC, LOGARITHMIC, NEGATIVEBINOMIAL, PASCAL, POISSON, UNIFORMINT]<br>gofType = [AndersonDarling, CramerVonMises, KolmogorovSmirnov, WatsonG, WatsonU] | PascalDist<br>NegativeBinomialDist<br>NegativeBinomialDist<br>NegativeBinomialDist<br>PascalDist | PascalDist<br>NegativeBinomialDist<br>NegativeBinomialDist<br>NegativeBinomialDist<br>PascalDist |
| 3 | Funcionalidad:<br>propagation(variables, exp, run, dataResult) | variables = [A, ANP, Sw]<br>exp = “A+ANP*Sw”<br>run = 1000 | ParetoDist<br>ParetoDist<br>ParetoDist<br>StudentDist<br>BetaSymmetricalDist | ParetoDist<br>ParetoDist<br>ParetoDist<br>StudentDist<br>BetaSymmetricalDist |

Tabla 9. Caso de Prueba de la biblioteca Simulate.

Las pruebas de unidad realizadas a la biblioteca Simulate fueron realizadas con éxito, comprobándose el correcto funcionamiento de cada una de las funcionalidades desarrolladas en la misma.

### 3.8. Pruebas de Caja Negra.

Con la finalidad de asegurar que el sistema está funcionando como fue establecido por los clientes y futuros usuarios del mismo, además de garantizar que aspectos específicos de su comportamiento tales como seguridad, rendimiento, fiabilidad y accesibilidad se cumplen satisfactoriamente, se diseñaron y ejecutaron pruebas de caja negra, en un entorno lo más parecido posible al entorno final de despliegue.

#### 3.8.1. Diseño de los Casos de Prueba.

Para la realización de las pruebas de sistema se utilizan como recursos físicos: un computador con microprocesador Intel Core 2 Duo de velocidad del CPU de 2.20 GHz, RAM de 1024 Mb y un disco duro con capacidad de 160 Gb; y como recursos lógicos: sistema operativo Windows 7 Ultimate y navegador web Firefox 12. Se realizaron 4 ciclos de pruebas para detectar y corregir errores que pueda presentar la aplicación.

- Caso de prueba: Introducir datos de campo de manera incorrecta.

| | |
|---|---|
| Caso de Uso | Caracterizar variables aleatorias. |
| Caso de prueba | Introducir datos de campo de manera incorrecta. |
| Entrada | El usuario introduce el valor 120a. |
| Resultado esperado | Mostrar un cartel indicando que debe colocar un valor numérico. |
| Resultado de la prueba | El sistema muestra un cartel indicando que debe colocar un valor numérico. |

Tabla 10. Caso de prueba: Introducir data de campo de manera incorrecta.

- Caso de prueba: Introducir data de campo de manera correcta.

| | |
|---|---|
| Caso de Uso | Caracterizar variables aleatorias. |
| Caso de prueba | Introducir data de campo de manera correcta. |
| Entrada | El usuario introduce el valor 120. |

| | |
|---|---|
| Resultado esperado | Se adiciona el valor introducido a la lista de valores de la variable a caracterizar. |
| Resultado de la prueba | Se adicionó el valor introducido al final de la lista de valores de la variable a caracterizar. |

Tabla 11. Caso de prueba: Introducir data de campo de manera correcta.

- Caso de prueba: Cambiar naturaleza de la variable.

| | |
|---|---|
| Caso de Uso | Caracterizar variables aleatorias. |
| Caso de prueba | Cambiar naturaleza de la variable. |
| Entrada | El usuario cambia el tipo de variable aleatoria de continua a discreta y selecciona aceptar. |
| Resultado esperado | Se muestra un mensaje informando que si cambia la naturaleza de la variable se borrarán todos los datos históricos de la misma. Al seleccionar la opción aceptar se borrarán los datos históricos de la variable y cambiarán las distribuciones de probabilidad continuas a las discretas. |
| Resultado de la prueba | Se mostró un cartel indicándole al usuario que si cambia la naturaleza de la variable se borrarán todos los datos históricos de la misma. Al seleccionar aceptar se borraron los datos históricos de la variable y cambiaron las distribuciones de probabilidad continuas a discretas. |

Tabla 12. Caso de prueba: Cambiar naturaleza de la variable.

- Caso de prueba: Caracterizar variables aleatorias por ajuste a un set de datos.

| | |
|---|---|
| Caso de Uso | Caracterizar variables aleatorias. |
| Caso de prueba | Caracterizar variables aleatorias por ajuste a un set de datos. |

| | |
|---|---|
| Entrada | El usuario secciona todas las pruebas de bondad de ajuste, todas las distribuciones de probabilidad, introduce los datos [430, 434, 446, 516, 597, 629, 647, 719, 737, 746] y presiona el botón “Caracterizar”. |
| Resultado esperado | Se muestra una interfaz con un gráfico mostrando el histograma de frecuencias de los valores introducidos y una tabla con el nivel de significancia de cada distribución de probabilidad seleccionada con respecto a las pruebas de bondad de ajuste elegidas. |
| Resultado de la prueba | Se mostró una interfaz con un gráfico mostrando el histograma de frecuencias de los valores introducidos y una tabla con el nivel de significancia de cada distribución de probabilidad seleccionada con respecto a las pruebas de bondad de ajuste elegidas. |

Tabla 13. Caso de prueba: Caracterizar variables aleatorias por ajuste a un set de datos.

- Caso de prueba: Introducir modelo matemático con errores sintácticos.

| | |
|---|---|
| Caso de Uso | Propagar incertidumbre. |
| Caso de prueba | Introducir modelo matemático con errores sintácticos. |
| Entrada | El usuario introduce el modelo matemático “A + ANP -”. |
| Resultado esperado | Se muestra un mensaje indicando que el modelo matemático contiene errores sintácticos. |
| Resultado de la prueba | Se mostró un mensaje indicando que el modelo matemático contiene errores sintácticos. |

Tabla 14. Caso de prueba: Propagar incertidumbre en un modelo matemático.

- Caso de prueba: Introducir modelo matemático con errores sintácticos.

| | |
|---|---|
| Caso de Uso | Propagar incertidumbre. |

| | |
|---|---|
| Caso de prueba | Propagar incertidumbre en un modelo matemático. |
| Entrada | El usuario introduce el modelo matemático “A + ANP * PNC” y llama al Caso de Prueba: Caracterizar variables aleatorias por ajuste a un set de datos. Luego presiona el botón propagar. |
| Resultado esperado | Se muestran las variables que contiene el modelo matemático y se realiza a cada una de las variables el Caso de Prueba: Caracterizar variables aleatorias por ajuste a un set de datos. Al seleccionar la opción “Propagar” se muestra una interfaz con un gráfico con el histograma de frecuencias de los valores aleatorios generados por el método de simulación de Montecarlo y una tabla con el nivel de significancia de cada distribución de probabilidad soportada con respecto a todas las pruebas de bondad de ajuste. |
| Resultado de la prueba | Se mostró una tabla con las variables que contiene el modelo matemático introducido y cada variable fue caracterizada correctamente. Al seleccionar la opción “Propagar” se mostró una interfaz con un gráfico con el histograma de frecuencias de los valores aleatorios generados por el método de simulación de Montecarlo y una tabla con el nivel de significancia cada distribución de probabilidad soportada respecto a todas las pruebas de bondad de ajuste. |

Tabla 15. Caso de prueba: Propagar incertidumbre en un modelo matemático.

Durante la ejecución del primer ciclo de pruebas se detectaron dos fallas que se corrigieron de inmediato por el equipo de desarrollo, en el resto de los ciclos de pruebas no se detectaron fallas. Actualmente el software no cuenta con fallos bloqueantes, ni críticos, por lo que cumple con los requisitos necesarios para su despliegue.

## 3.9. Conclusiones.

Esta etapa de desarrollo se caracteriza por resultados ya visibles para los clientes y gratificantes para los desarrolladores, ya que queda implementada la aplicación con las principales funcionalidades que se definieron para la iteración del producto. Además se llevó a cabo una importante tarea: las pruebas, realizadas al sistema ya como un producto ejecutable, que aunque no aseguran la no existencia de fallos o errores, si dan una alta confiabilidad y calidad a la aplicación.

## CONCLUSIONES GENERALES

Con el desarrollo de la investigación se obtuvieron resultados satisfactorios por lo que se le dio cumplimiento al objetivo planteado: Desarrollar un módulo de simulación de Montecarlo para el Sistema Integral de Confiabilidad.

Se diseñó e implementó un módulo de simulación de Montecarlo multiplataforma, de código abierto y bajo los principios de la soberanía tecnológica con soporte a un gran número de distribuciones de probabilidad, pruebas de bondad de ajuste, algoritmos para la caracterización probabilística de variables aleatorias y algoritmos para la propagación de la incertidumbre a cada variable aleatoria en los modelos matemáticos. Se integró el módulo de simulación de Montecarlo a la aplicación del proyecto SIC. Se diseñó e implementó una suite de pruebas de unidad para la validación la solución. Se documentó y estandarizó el código fuente de la aplicación.

RECOMENDACIONES

Cada día los ingenieros en confiabilidad requieren de nuevas funcionalidades. Por lo que se recomienda:

- Implementar la prueba de bondad de ajuste Chi Cuadrado a la biblioteca Simulate.
- Implementar la funcionalidad de correlacionar las variables aleatorias presentes en los modelos matemáticos con el fin de obtener resultados con mayor realismo en la propagación de la incertidumbre.

REFERENCIA BIBLIOGRÁFICA

1. **Yañez M., Medardo, y otros, y otros.** *Confiabilidad Integral.* Primera edición. Maracaibo, Zulia : RELIABILITY AND RISK MANAGEMENT S.A (R2M S.A), 2007. pág. 305. Vol. I.

2. **Yañez M., Medardo, Gómez de la Vega, Hernando y Valbuena, Genebelin.** *Gerencia de la Incertidumbre.* Maracaibo, Zulia : Casa Editorial Campus MBA Venezuela, Junio de 2003.

3. **Yañez, Medardo, Semeco, Karina y Medina, Nayrih.** *Enfoque Práctico para la Estimación de Confiabilidad y Disponibilidad.* Maracaibo : s.n., 2005.

4. Java. [En línea] [Citado el: 28 de Mayo de 2012.] http://www.java.com/es/about/.

5. groovy. [En línea] [Citado el: 28 de Mayo de 2012.] http://docs.codehaus.org/display/GROOVY/Home.

6. escueladegroovy. [En línea] [Citado el: 28 de Mayo de 2012.] http://www.escueladegroovy.com/informacion/groovy-grails.

7. **Google.** Google Code. [En línea] Google, 2011. [Citado el: 15 de Marzo de 2012.] http://code.google.com.es.mk.gd/webtoolkit/.

8. **Rocher, Graeme, y otros, y otros.** The Grails Framework - Reference Documentation. [En línea] [Citado el: 28 de Mayo de 2012.] http://grails.org/doc/latest/guide.

9. **SSJ.** SSJ: Stochastic Simulation in Java. [En línea] [Citado el: 10 de Mayo de 2012.] http://www.iro.umontreal.ca/~simardr/ssj/indexe.html.

10. Eclipse. [En línea] [Citado el: 28 de Mayo de 2012.] http://www.eclipse.org/org/.

11. **UML.** Introduction to OMG's Unified Modeling Language. [En línea] [Citado el: 20 de Mayo de 2012.] http://www.omg.org/gettingstarted/what_is_uml.htm.

12. **Visual Paradigm.** [En línea] [Citado el: 5 de Mayo de 2012.] http://www.visual-paradigm.com/product/vpuml/.

13. **IBM.** IBM Rational Unified Process (RUP). [En línea] [Citado el: 10 de Mayo de 2012.] http://www-01.ibm.com/software/awdtools/rup/.

14. **Pressman, Roger.** *Software Engineering, A Practitioner´s Approach.* New York : McGraw-Hill, 2010.

15. **Terreros, Julio Casal.** Desarrollo de Software basado en Componentes. *Microsoft.* [En línea] [Citado el: 29 de Mayo de 2012.] http://msdn.microsoft.com/es-es/library/bb972268.aspx#ref08.

16. **De La Torre Llorente, Cesar, y otros, y otros.** *Guia de Arquitectura N-Capas orientada al Dominio com.NET 4.0.* España : Krasis Consulting, 2010.

17. **Encyclopædia Britannica Inc.** client-server architecture. [En línea] [Citado el: 25 de Marzo de 2012.] http://www.britannica.com/EBchecked/topic/1366374/client-server-architecture.

18. **Microsoft.** Model-View-Presenter Pattern. [En línea] [Citado el: 5 de Mayo de 2012.] http://msdn.microsoft.com/en-us/library/ff647543.aspx.

19. **Sun Microsystems, Inc.** *Java Code Conventions.* Mountain View, California : s.n., 1999.

20. **Oracle.** Oracle Crystal Ball. [En línea] [Citado el: 10 de Febrero de 2012.] www.oracle.com/us/products/applications/crystalball/index.html.

21. **@Risk.** Risk. [En línea] [Citado el: 10 de Marzo de 2012.] http://www.palisade-lta.com/risk/.

BIBLIOGRAFÍA


1. **Yañez M., Medardo, et al., et al.** *Confiabilidad Integral.* Primera edición. Maracaibo, Zulia : RELIABILITY AND RISK MANAGEMENT S.A (R2M S.A), 2007. p. 305. Vol. I.
2. **Yañez M., Medardo, Gómez de la Vega, Hernando and Valbuena, Genebelin.** *Gerencia de la Incertidumbre.* Maracaibo, Zulia : Casa Editorial Campus MBA Venezuela, Junio de 2003.
3. **Yañez, Medardo, Semeco, Karina and Medina, Nayrih.** *Enfoque Práctico para la Estimación de Confiabilidad y Disponibilidad.* Maracaibo : s.n., 2005.
4. groovy. [Online] [Cited: Mayo 28, 2012.] http://docs.codehaus.org/display/GROOVY/Home.
5. **Google.** Google Code. [Online] Google, 2011. [Cited: Marzo 15, 2012.] http://code.google.com.es.mk.gd/webtoolkit/.
6. **Rocher, Graeme, et al., et al.** The Grails Framework - Reference Documentation. [Online] [Cited: Mayo 28, 2012.] http://grails.org/doc/latest/guide.
7. **SSJ.** SSJ: Stochastic Simulation in Java. [Online] [Cited: Mayo 10, 2012.] http://www.iro.umontreal.ca/~simardr/ssj/indexe.html.
8. Eclipse. [Online] [Cited: Mayo 28, 2012.] http://www.eclipse.org/org/.
9. **UML.** Introduction to OMG's Unified Modeling Language. [Online] [Cited: Mayo 20, 2012.] http://www.omg.org/gettingstarted/what_is_uml.htm.
10. **IBM.** IBM Rational Unified Process (RUP). [Online] [Cited: Mayo 10, 2012.] http://www-01.ibm.com/software/awdtools/rup/.
11. **Pressman, Roger.** *Software Engineering, A Practitioner´s Approach.* New York : McGraw-Hill, 2010.
12. **Oracle.** Oracle Crystal Ball. [Online] [Cited: Febrero 10, 2012.] www.oracle.com/us/products/applications/crystalball/index.html.
13. **@Risk.** Risk. [Online] [Cited: Marzo 10, 2012.] http://www.palisade-lta.com/risk/.

GLOSARIO DE TÉRMINOS

**Caracterización probabilística**: determinar los atributos probabilísticos peculiares de alguien o de algo, de modo que claramente se distinga de los demás.

**Código abierto**: es una tendencia internacional del desarrollo de software que profesa la distribución del código junto a las aplicaciones, se rigen por licencias tales como GNU/GPL.

**Confiabilidad**: es la probabilidad de que un componente, sistema o proceso realice en un intervalo de tiempo determinado lo que se requiere del mismo.

**Incertidumbre**: grado de desconocimiento de un tema.

**Modelos matemáticos**: representación conceptual empleada para concretar la expresión de vínculos entre variables, operaciones u otra clase de entidades a través de una fórmula matemática.

**Multiplataforma**: es un término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

**Probabilidad**: En un proceso aleatorio, razón entre el número de casos favorables y el número de casos posibles.

**RUP**: Rational Unified Project (Proceso Unificado de Desarrollo).

**SIC**: Sistema Integral de Confiabilidad.

**Simulación**: Alteración aparente de la causa, la índole o el objeto verdadero de un acto o contrato.

**Simulación de Montecarlo**: La simulación a través del algoritmo matemático de Montecarlo.

**Subsistema**: Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos.

**UCI**: Universidad de las Ciencias Informáticas.

**UML**: Unified Modeling Language (Lenguaje Unificado de Modelado): Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

**Variable aleatoria**: Magnitud que puede tener un valor cualquiera de los comprendidos en un conjunto.