Implementación de autómatas celulares usando desarrollo dirigido por comportamiento

Implementation of cellular automata using behavior-driven development

Victor Vargas-Forero[1]
Luz Muñoz-Ceballos[2]
Ángel García-Baños[3]

[1] Universidad del Valle (Colombia). Correo electrónico: victor.m.vargas@correounivalle.edu.co; orcid: http://orcid.org/0000-0002-2317-7365

[2] Universidad del Valle (Colombia). Correo electrónico: luz.estela.munoz@correounivalle.edu.co; orcid: http://orcid.org/0000-0002-2675-1238

[3] Universidad del Valle (Colombia). Correo electrónico: angel.garcia@correounivalle.edu.co; orcid: http://orcid.org/0000-0002-5388-6912

Recibido: 15-11-2018 Aceptado: 06-08-2019

Cómo citar: Vargas-Forero, Victor; Muñoz-Ceballos, Luz; García-Baños, Ángel (2020). Implementación de autómatas celulares usando desarrollo dirigido por comportamiento. *Informador Técnico, 84*(1), 48-66. https://doi.org/10.23850/22565035.2257

Resumen

En este trabajo se presenta parte de la implementación de un autómata celular usando el desarrollo dirigido por comportamiento Behavior-Driven Development (BDD) como estrategia para la construcción de una herramienta que permita modelar la accesibilidad peatonal dentro de un espacio abierto como es el campus de la Universidad del Valle en Cali, Colombia. Para la implementación se usó BDD como método de desarrollo, Python como lenguaje de programación y Behave para escribir las pruebas en lenguaje natural. Como resultados se presentaron parte del modelo implementado del autómata celular y algunos de los requerimientos con sus respectivos escenarios para BDD e implementación, donde se explicó el uso del framework Behave en Python. Se concluyó que BDD es una buena herramienta para este tipo de proyectos porque se obtuvo como resultado 706 líneas de código fuente correspondientes al modelo del autómata celular, probadas con un total de 144 pruebas definidas desde la fase inicial del proyecto y superadas satisfactoriamente, de acuerdo a una encuesta de aceptación del uso de la metodología BDD por parte de los usuarios del proyecto, y la primera aplicación de BDD en una simulación de movilidad peatonal.

Palabras clave: software ágil; BDD; autómatas celulares; accesibilidad peatonal; procesos y métodos de desarrollo de software.

Abstract

This work aimed to present part of the implementation of a cellular automaton using Behavior-Driven-Development (BDD) as a strategy for the construction of a tool that allows to model pedestrian accessibility within an open space such as the campus of the Universidad del Valle in Cali, Colombia. For the implementation, BDD was used as a development method, Python as a programming language and Behave to write the tests in natural language. As a result, part of the implemented model of the cellular automaton was presented, some of the requirements with their respective scenarios for BDD and implementation, where the use of the Behave framework in Python was explained. It was concluded that BDD is a good tool for this type of projects because it resulted in 706 lines of source code corresponding to the model of the cellular automaton

tested with a total of 144 tests defined from the initial phase of the project and successfully passed, a survey of acceptance of the use of the BDD methodology by project users and the first application of BDD in a pedestrian mobility simulation.

Keywords: agile software; BDD, cellular automata; pedestrian accessibility; processes and software development methods.

1. Introducción

Se presenta la experiencia de una implementación del modelo de un autómata celular para analizar la accesibilidad peatonal al interior del campus universitario de la ciudadela de Meléndez de la Universidad del Valle, tomando como punto de partida el estudio realizado por Buitrago y Kattán (2011), además de implementar el desarrollo dirigido por el comportamiento *Behavior-driven development* (BDD) (North, 2006), como herramienta metodológica para dicha construcción, a partir del resultado del trabajo de grado de Muñoz, 2017).

Mediante el autómata celular se construyó el núcleo del simulador de los peatones del campus universitario, modelo que fue presentado en Vargas-Forero; Muñoz-Ceballos; García-Baños; Jaramillo-Molina (2019). Existen diversos estudios que se han centrado en problemas de microsimulación, como por ejemplo (Gipps; Marksjö, 1985), pero la mayoría de estos problemas se centran en la evacuación de los peatones (Gudowski; Waș, 2007) y algunos involucran las características ambientales y los comportamientos de los vecinos (Was, 2005). Los problemas de accesibilidad y movilidad peatonal son actualmente un campo muy analizado, como se estudia en Martins-Gonçalves; Nuñez-Basantes (2017); Chen *et al*., (2018); Qin; Curtin; Rice (2018).

Por otra parte, otro reto para el proyecto del grupo de trabajo era probar el desarrollo de software ágil en un proyecto de investigación usando autómatas celulares. Desde el 2001 con la iniciativa del Manifiesto Ágil (Beck, 2001) inició una corriente muy fuerte que busca cambiar la forma como de desarrollar software (metodologías tradicionales, la más representativa para la época RUP (Khamis; Abdelmonem, 2002). Como se puede observar en Poppendieck y Cusumano (2012), existe una gran diferencia entre los procesos de desarrollo ágil y las metodologías tradicionales, por tal motivo, se propone un conjunto de principios para guiar el proceso de desarrollo. A partir del Manifiesto Ágil, surgen muchas iniciativas como Test Driven Development (TDD), Acceptance Test Driven Development (ATDD), Behavior Driven Development (BDD), entre otras. Se decide trabajar con BDD por las características que presenta este proceso de desarrollo de software como lenguaje ubicuo (basado en el dominio del problema), proceso de descomposición iterativa, descripción de texto con historias de usuario y plantillas de escenarios, pruebas de aceptación automatizadas con reglas de mapeo, código de especificación orientado al comportamiento legible y diferentes fases del desarrollo dirigidas por el comportamiento. Además, se optó por Behave (Engel; Rice; Jones, 2012) como herramienta de soporte para BDD, por ser una de las herramientas que más características cumple (Solis; Wang, 2011).

Por lo anterior, esta investigación se desarrolla de la siguiente manera: en la primera parte se expone la metodología propuesta. En la segunda se hace una breve presentación del modelo del autómata celular a construir. En la tercera se muestran algunos de los resultados obtenidos al construir el modelo usando BDD y al final, en la cuarta parte, se presentan las conclusiones del desarrollo realizado.

2. Metodología

El trabajo se desarrolló siguiendo las directrices de *BDD* (North, 2006; Ferguson-Smart, 2014), estas se centran en los "comportamientos" esperados del software y no en cómo debe ser implementada la lógica del mismo, facilitando el entendimiento entre el usuario final y el desarrollador. Esta característica era deseable en el proyecto porque se esperaba que los requerimientos cambiasen muy frecuentemente a lo largo de la ejecución del mismo. BDD surge desde la iniciativa del Manifiesto Ágil (Beck *et al.*, 2001), la cual se fundamenta en los valores indicados en la Tabla 1, dando mayor valor a los elementos de la izquierda que a los de la derecha:

Tabla 1.
BDD del Manifiesto Ágil

Izquierda		Derecha
Individuos e interacciones	sobre	Procesos y herramientas
Software funcionando	sobre	Documentación extensiva
Colaboración con el cliente	sobre	Negociación contractual
Respuesta ante el cambio	sobre	Seguir un plan

Fuente: elaboración propia.

A partir de esta iniciativa surgen otras, que buscan mejorar las metodologías tradicionales que se usaban hasta la fecha, una de estas es TDD (Jurado, 2010) que involucra otras dos prácticas: escribir las pruebas antes de escribir el código del programa y refactorizar hasta terminar de construir toda la solución. BDD se puede ver como una mejora de TDD, donde las características de TDD son incorporadas en BDD, de tal manera que TDD se centra en las pruebas unitarias y BDD se enfoca en pruebas de más alto nivel, en la funcionalidad, la aceptación, la búsqueda de definir el problema y no como se debe resolver.

En la Figura 1 se observa que la estrategia metodológica en general para BDD, consiste en construir pruebas de aceptación del usuario, pruebas unitarias, luego hacer el código necesario para pasar dichas pruebas y refactorizar con las nuevas pruebas, realizando este proceso en ciclos e incorporando las nuevas funcionalidades.

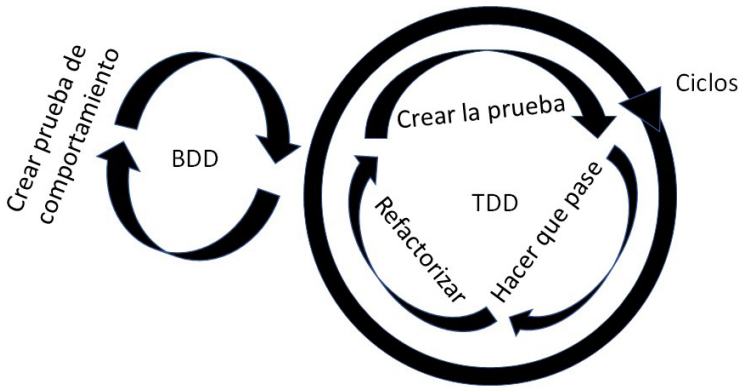

Figura 1. Desarrollo dirigido por el comportamiento
Fuente: elaboración propia.

3. Modelo

A continuación, se presenta brevemente el modelo del autómata celular implementado. Para profundizar en este, se recomienda leer la definición completa en (Vargas-Forero *et al.*, 2019). El autómata celular se definió como una tupla (G, S, N, R), es decir, un conjunto ordenado de objetos caracterizados por:

- (G): la rejilla
- (S): los estados que pueden tomar una celda
- (N): la vecindad
- (R): las reglas de transición

Para el proyecto de la accesibilidad peatonal con autómatas celulares se definió:

3.1. (G) Rejilla o cuadrícula, donde cada celda puede contener un peatón

• Cada celda es cuadrada y mide aproximadamente 40 x 40 , valor tomado por sugerencia en (Schadschneider, 2001).

• El número de peatones que puede haber en una celda es 0 o 1.

• El autómata se define sobre una cuadrícula que representa el campus universitario y cuyas dimensiones son 2.589 filas por 1.906 columnas, que se obtiene al dividir la dimensión de todo el campus universitario con el tamaño de la celda propuesta.

3.2. (S) Conjunto finito de estados que puede tomar una celda

Un elemento adicional tenido en cuenta en esta definición del modelo, es el concepto de impedancia y será entendida como la resistencia (disminución de velocidad) que tiene un peatón al desplazarse por una vía o camino, ocasionada por motivos de una rampa, escalera o dificultad de pasar por la vía.

Para el modelo propuesto, las ("") representan una celda vacía equivalente a un sendero no transitable, la letra P representa una celda ocupada por un peatón, el número 0 una celda vacía de un sendero transitable sin impedancia y cuando la celda tenga un valor entre 1 y n, esto representa que la celda está vacía y el sendero es transitable con impedancia. El valor de la impedancia depende del valor de n, en el que 1 es la impedancia mínima que retrasa al peatón un ciclo por cada movimiento, 2 retrasaría el peatón dos ciclos por cada movimiento y así sucesivamente.

3.3. (N) Vecindad correspondiente a cada celda

Para el proyecto se tomó como base la vecindad de Moore extendida que es una extensión de la vecindad de Von Newman. Esta vecindad contiene celdas en la diagonal como se presenta en la Figura 2. Para entender un poco más el concepto ver Wolfram (2019).

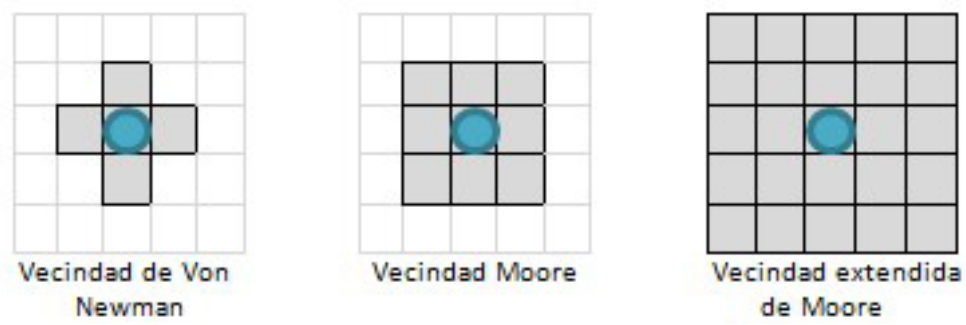

Figura 2. Tipos de vecindad
Fuente: elaboración propia.

3.4. (R) Reglas de transición que definen el modelo del autómata celular

La propuesta del modelo tiene el alcance de la vecindad máxima a 2 celdas y no se consideran todos los vecinos dentro de los movimientos posibles. El peatón no considera retroceder en su trayectoria de origen a destino, por lo que solo realiza movimientos en un solo sentido con dos velocidades. Como se puede ver en la Figura 3, en el lado izquierdo se observa que las flechas solo se desplazan una posición y apuntan solo en la dirección de origen y destino, y no apuntan del destino al origen, esto indica que los peatones no retroceden en el modelo y solo se mueven hacia adelante una posición que implica a la velocidad 1. En el lado derecho se puede observar que es parecido al de la izquierda, solo que el movimiento es de dos celdas, esto corresponde a la velocidad 2.

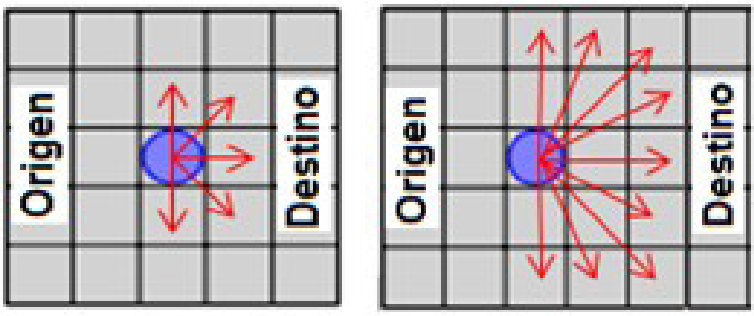

Figura 3. Tipos de velocidad, izquierda velocidad 1 y derecha velocidad 2
Fuente: elaboración propia.

Tras lo anterior, solo se presenta una idea base para entender algunos elementos de la implementación, con el fin de entender con mayor profundidad este modelo se recomienda ver el artículo de Vargas-Forero *et al.* (2019) donde se presenta cómo se mueve el peatón en el autómata celular.

4. Resultados

A continuación, se presentan, explican y se muestran algunos ejemplos de las etapas contempladas en el proceso de construcción de dicho desarrollo:

4.1. Planeación

BDD es una técnica de desarrollo ágil orientada al negocio y comportamiento de una aplicación. Dado que su principal objetivo es preguntar qué debería hacer la aplicación, se utilizaron técnicas de indagación con los usuarios finales, de manera que éstas permitieran formular y escribir las pruebas (tests) de comportamiento. Con el propósito de dar claridad al desarrollo, se realizaron prototipos que dieran solución a uno o varios de los requerimientos. Para que el desarrollo se diera de forma incremental, se decidió trabajar desde los requerimientos más básicos a los más complejos.

4.2. Requerimientos del sistema

Como apoyo al proceso de planeación se definió, de forma preliminar, una especificación de requerimientos para establecer el alcance de la aplicación. Con las necesidades determinadas por los requerimientos se delimitó el alcance de la aplicación y se estimó una aproximación clara de las funcionalidades para dar cumplimiento al objetivo propuesto. La solución debe contar con las siguientes funcionalidades:

- Representar la red peatonal del campus universitario Sede Meléndez.
- Representar peatones.
- Generar peatones.
- Movimiento de los peatones.
- Implementación de las reglas del modelo del autómata celular.
- Verificación de las reglas del autómata celular.

• Interfaz gráfica que permita observar el comportamiento de los peatones sobre la red peatonal del campus universitario Sede Meléndez.

• El sistema debe funcionar sobre un sistema de información geográfico.

4.3. Especificación de los requerimientos

• El sistema debe permitir representar la red peatonal del campus universitario. Esta red debe contemplar las 60 edificaciones y la red peatonal (rutas) correspondiente a la cartografía suministrada por Grupo de Investigación en Transporte, Tránsito y Vías – GITTV.

• El sistema debe permitir representar un peatón con las siguientes características: origen, destino, ruta, velocidad y posición actual en la cual está ubicado el peatón.

• El sistema debe permitir generar peatones para cada ruta dado un origen y un destino, el tiempo inicial y tiempo final de generación de los peatones, el número de peatones a generar, la probabilidad de generación de los peatones y la probabilidad para generar peatones de velocidad 1 y 2 como se puede observar en la Figura 3 del modelo.

Movimiento de los peatones:

• El sistema debe proporcionar un mecanismo para simular el movimiento de los peatones.

• El sistema debe proporcionar un mecanismo para resolver los obstáculos.

• El sistema debe permitir configurar segmentos de rutas para evaluar el flujo peatonal.

• Dadas las coordenadas de un segmento de la ruta, el sistema debe determinar cuántos peatones pasan en un tiempo T.

• El sistema debe permitir configurar segmentos de rutas para evaluar la densidad peatonal.

• Dadas las coordenadas de un segmento de la ruta, el sistema debe determinar cuántos peatones hay en un tiempo T.

• Dada la ubicación de un peatón en la rejilla que representa el campus universitario Sede Meléndez, el sistema debe determinar si es posible que un peatón se mueva y cuál es su nueva ubicación.

• Implementación de las reglas definidas para el modelo del autómata celular.

Verificación de las reglas del modelo del autómata celular:

• El sistema debe permitir calcular la distancia del peatón a su lugar de destino.

• Dada el lugar de origen y lugar de destino de un peatón, el sistema debe determinar la orientación de este.

• Dada la ubicación actual de un peatón, el sistema debe determinar su próxima ubicación y verificar si ese movimiento está fuera del rango establecido.

• Dada la ubicación actual de un peatón, el sistema debe determinar su próxima ubicación y verificar si ese movimiento está fuera de la ruta establecida.

• Dada la ubicación actual de un peatón, el sistema debe determinar su próxima ubicación y verificar si ese movimiento está fuera de la rejilla establecida.

• Dada un peatón que se encuentra en una posición actual y cuya velocidad es 2, el sistema debe evaluar si se puede desplazar y a qué velocidad (velocidad 2, velocidad 1, no se mueve).

• Dado un peatón que se encuentra en una posición actual y cuya velocidad es 1, el sistema debe evaluar si se puede desplazar (velocidad 1, no se mueve).

• El sistema debe proporcionar una interfaz gráfica que permita observar el comportamiento de los peatones sobre la red peatonal del campus universitario ciudad Meléndez.

• Una vez inicia la ejecución del sistema, este debe mostrar la cuadrícula que representa el campus universitario sede Meléndez y la red peatonal.

• Dadas las coordenadas esquina superior izquierda y esquina inferior derecha, el sistema debe mostrar el escenario correspondiente a dichas coordenadas en la cuadrícula que representa el campus universitario sede Meléndez.

• El sistema debe proporcionar un menú emergente, mediante el cual el usuario podrá seleccionar diferentes escenarios previamente establecidos para observar el comportamiento de los peatones.

• Mostrar los movimientos de los peatones.

• El sistema debe verificar la ubicación de todos los peatones y actualizar la interfaz, mostrando de esta manera cómo fluyen los peatones dado un origen y un destino dentro de la red peatonal que representa el campus universitario Sede Meléndez.

4.4. Uso de BDD

Lo primero que se debe definir en BDD es la creación de los escenarios del proyecto, es decir, los requerimientos del proyecto, donde se define cada una de sus necesidades. Se toma como ejemplo el escenario peatón fuera del rango, ya que es un buen ejemplo por ser fácil de entender y sirve para explicar el uso de BDD. En la Tabla 1 se presentan algunos ejemplos del uso de BDD para el autómata celular:

Tabla 2.
Plantilla BDD. Verificar si el movimiento está fuera del rango de la matriz

Parte	Descripción
Scenario Outline	Verifica si el movimiento está fuera del rango de la matriz.
Given (*Dado*)	El inicio a la simulación, se definen las reglas del autómata celular.
When (*Cuando*)	Cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (<*matriz*>) donde se puede mover el peatón, la posición <*actual*> del peatón y los valores de movimiento en <*movimientoxy*>.
Then (*Entonces*)	Se determina que el peatón en la posición <*actual*> con movimiento en <*movimientoxy*>, tiene (<*resultado*>) donde 1 es un movimiento fuera del rango y 0 no.

Fuente: elaboración propia.

Como se puede ver en la Tabla 2, el escenario se presenta en cuatro partes las cuales se explican a continuación:

• **Scenario Outline**: se presenta una descripción de lo que debe hacer esta funcionalidad del programa. El caso verifica si el movimiento está fuera del rango de la matriz de peatones. Eso quiere decir que deberá retornar verdadero si el movimiento está fuera de la matriz.

• **Given**: se determinan los elementos iniciales necesarios para que la funcionalidad se pueda implementar. Para el escenario, se inicia una instancia del objeto reglas. Este objeto reglas se crea al iniciar la simulación.

• **When**: en este campo se definió el momento y los elementos necesarios para realizar la funcionalidad. Para este caso se definió una matriz donde se puede mover el peatón, la posición donde el peatón se encuentra y, por último, se necesita la posición donde el peatón será movido.

• **Then**: finalemente, se definió el resultado esperado de la funcionalidad. Para el ejemplo se utilizó el resultado con la posición dada del peatón y la nueva posición (x, y) del mismo.

Para completar el escenario se proponen las pruebas que deberá pasar la implementación.

Como se puede observar en la Figura 4, se presentan todos los ejemplos a realizar para entender y probar la funcionalidad. A continuación, se realizará una explicación de cada uno de ellos la primera característica es la verificación si un movimiento está dentro del rango posible de un peatón.

Ejemplos: fuera_del_rango_matriz

matriz	actual	movimientoxy	resultado
0,0,0,1,1,0,1,1	0,0	0,0	0
0,0,0,1,1,0,1,1	0,0	1,0	0
0,0,0,1,1,0,1,1	1,1	1,0	1
0,0,0,1,1,0,1,1	0,0	2,2	1
0,0,0,1,1,0,1,1	0,0	1,1	0

Figura 4. Ejemplos escenario 1
Fuente: elaboración propia.

Escenario: movimiento posible del peatón.

• **Dado**: el movimiento de un peatón de una posición origen a una posición destino y todos los movimientos posibles.

• **Cuando**: la posición destino sea un movimiento posible.

• **Entonces**: la nueva posición del peatón es la posición destino.

Escenario: movimiento no posible del peatón.

• **Dado**: el movimiento de un peatón de una posición origen a una posición destino y todos los movimientos posibles.

• **Cuando**: la posición destino no sea un movimiento posible.

• **Entonces**: la nueva posición es la misma posición actual.

Hasta aquí termina la primera parte de BDD, donde se definieron las funcionalidades que necesitan ser desarrolladas para resolver el problema y completar la solución. El siguiente paso fue construir las pruebas necesarias desde la más elemental hasta la más compleja para construir el programa.

A continuación, se presenta la prueba que fue desarrollada para dar la solución del problema. La primera parte define el uso de las librerías que se necesitan en Python para el uso de BDD, como también la definición del tag @given, en el cual se inicia la instancia de las reglas del autómata celular para definir sus movimientos.

```
from behave   import given, when, then
from hamcrest import assert_that, equal_to, is_not
from Reglas import Reglas
@given('se definen las reglas del autómata celular')
def step_inicia(context):
    context.objReglas = Reglas()
```

En esta parte de la prueba se define el tag @when. En esta etiqueta se realizó la toma de los datos de la prueba, como también la ejecución de los programas construidos para realizar la o las funcionalidades definidas en los escenarios.

@when('se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida.

Como ejemplo, para la matriz **(**{matriz}**)** donde se mueve el peatón, la posición {actual} **del** peatón y los valores de movimiento en **(**{movimientoxy}**).**')

```
def step_fueraDelRango(context, matriz, actual, movimientoxy):

    mat = matriz.split(',')

    sig = 0

    dic = {}

    tam = int(len(mat)/2)

    dic['tamano'] = ( int(tam/2), int(tam/2) )

    for x in range (0, tam):

            dic[(int(mat[ sig ]), int(mat[ sig + 1 ]))] = 0

            sig += 2

    x1 = int(actual.split(',')[0])

    y1 = int(actual.split(',')[1])

    x = int(movimientoxy.split(',')[0])

    y = int(movimientoxy.split(',')[1])

    context.resultadoMovimiento =  context.objReglas.

    fueraDelRango( dic , [x1, y1], x, y )
```

Como se observa en el código anterior, la mayor parte es para construir la prueba con los datos definidos en el escenario y solo se usó una sola línea de código para llamar la funcionalidad que verifica si el movimiento del peatón se encuentra dentro de la matriz definida para el movimiento de este.

Por último, se definió la comparación de los datos de prueba. En este código se observó que solo se evalúa si el resultado de la ejecución del código es igual al esperado en el escenario definido previamente.

```
@then('se determina que el peatón en la posición {actual}

con movimiento en ({movimientoxy}), tiene ({resultado})

donde 1 es un movimiento fuera del rango y 0 no.')

def step_verifica_movimiento(context, actual, movimientoxy, resultado):

    assert_that(bool(int(resultado)),

    equal_to(context.resultadoMovimiento))
```

Esta prueba se aplicó para cada uno de los datos definidos y se obtiene como resultado la siguiente salida por consola a la ejecución de las pruebas.

• **Scenario Outline**: verifica si el movimiento está fuera del rango de la matriz. -- @1.1 fuera_del_rango_matriz # peatones.feature:31.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (0,0,0,1,1,0,1,1) donde se mueve el peatón, la posición 0,0 del peatón y los valores de movimiento en (0,0). \# steps\step_reglas.py:34.

• **Then**: se determina que el peatón en la posición 0,0 con movimiento en (0,0), tiene (0) donde 1 es un movimiento fuera del rango y 0 no. # steps\step_reglas.py:50.

• **Scenario Outline**: verifica si el movimiento está fuera del rango de la matriz. -- @1.2 fuera_del_rango_matriz # peatones.feature:32.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (0,0,0,1,1,0,1,1) donde se mueve el peatón, la posición 0,0 del peatón y los valores de movimiento en (1,0). # steps\step_reglas.py:34.

• **Then**: se determina que el peatón en la posición 0,0 con movimiento en (1,0), tiene (0) donde 1 es un movimiento fuera del rango y 0 no. # steps\step_reglas.py:50.

• **Scenario Outline**: verifica si el movimiento está fuera del rango de la matriz. -- @1.3 fuera_del_rango_matriz # peatones.feature:33.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (0,0,0,1,1,0,1,1) donde se mueve el peatón, la posición 1,1 del peatón y los valores de movimiento en (1,0). # steps\step_reglas.py:34.

• **Then**: se determina que el peatón en la posición 1,1 con movimiento en (1,0), tiene (1) donde 1 es un movimiento fuera del rango y 0 no. # steps\step_reglas.py:50.

• **Scenario Outline**: verifica si el movimiento está fuera del rango de la matriz. -- @1.4 fuera_del_rango_matriz # peatones.feature:34.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (0,0,0,1,1,0,1,1) donde se mueve el peatón, la posición 0,0 del peatón y los valores de movimiento en (2,2). # steps\step_reglas.py:34.

• **Then**: se determina que el peatón en la posición 0,0 con movimiento en (2,2), tiene (1) donde 1 es un movimiento fuera del rango y 0 no. # steps\step_reglas.py:50.

• **Scenario Outline**: verifica si el movimiento está fuera del rango de la matriz. -- @1.5 fuera_del_rango_matriz # peatones.feature:35.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento está dentro de la matriz definida. Como ejemplo, para la matriz (0,0,0,1,1,0,1,1) donde se mueve el peatón, la posición 0,0 del peatón y los valores de movimiento en (1,1). # steps\step_reglas.py:34.

• **Then**: se determina que el peatón en la posición 0,0 con movimiento en (1,1), tiene (0) donde 1 es un movimiento fuera del rango y 0 no. # steps\step_reglas.py:50.

En los datos expuestos se observa la ejecución de cada una de las pruebas propuestas, y se puede ver que todas las pruebas pasaron correctamente. Para ejecutar esta prueba se implementó en Python el siguiente código:

```
import Peaton

class Reglas:  # Reglas de transformación del autómata.

#  fueraDelRango: retorna true si los valores están fuera del rango de la matriz.

#  param: matriz, posición del peatón, movimiento de “x” y “y” para la nueva posición.

#  return: verdadero si el movimiento es inválido.

def fueraDelRango(self, matriz, posicion, x, y):

    # Valida que el movimiento se sale del rango de la matriz

    if ((posicion[0]+x>(matriz[“tamano”][0]-1))

     or ((posicion[0]+x)<0)

     or (posicion[1]+y>(matriz[“tamano”][1]-1))

     or ((posicion[1]+y)<0)):

            return True return False
```

Se pudo observar que este código es bastante simple y se seleccionó con ese propósito para simplificar esta parte y explicar su implementación de BDD. A continuación, se presentan otros dos escenarios con su respectiva ejecución, los cuales son relevantes para las reglas del autómata celular.

4.5. Ejecución del escenario que verifica el avance de una celda del peatón

Se presentan los resultados de la ejecución de la prueba en BDD de la Tabla 3 y los ejemplos definidos en la Figura 5, para el escenario donde se verifica el avance de una sola celda del peatón.

Tabla 3.
Plantilla BDD. Avance de una sola celda del peatón

Parte	Descripción
Scenario Outline	Avanza un peatón una sola celda hacia adelante, si puede.
Given (*Dado*)	Se definen las reglas del autómata celular.
When (*Cuando*)	Cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón <destino> y la posición <actual> del peatón para avanzar una posición.
Then (*Entonces*)	Se determina que el peatón en la posición <actual>, tiene <resultado> de movimiento en el camino para un paso.

Fuente: elaboración propia.

Ejemplos: adelante_uno_movimiento

destino	actual	resultado
4,4	2,2	1,1
0,0	2,2	-1,-1
0,4	2,2	-1,1
4,0	2,2	1,-1
2,0	2,2	0,-1
2,5	2,2	0,1
0,2	2,2	-1,0
5,2	2,2	1,0

Figura 5. Ejemplos escenario de avance de una sola celda del peatón
Fuente: elaboración propia.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.1 adelante_uno_movimiento # peatones.feature:67.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

When: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 4,4 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (1,1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.2 adelante_uno_movimiento # peatones.feature:68.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,0 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-1,-1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.3 adelante_uno_movimiento # peatones.feature:69.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,4 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-1,1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.4 adelante_uno_movimiento # peatones.feature:70.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 4,0 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (1,-1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.5 adelante_uno_movimiento # peatones.feature:71.

• **Given**: se definen las reglas del autómata celular # steps\textbackslash step_ reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 2,0 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (0,-1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.6 adelante_uno_movimiento # peatones.feature:72.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 2,5 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (0,1) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.7 adelante_uno_movimiento # peatones.feature:73.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,2 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-1,0) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

• **Scenario Outline**: avanza un peatón una sola celda hacia adelante, si puede. -- @1.8 adelante_uno_movimiento # peatones.feature:74.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 5,2 y la posición 2,2 del peatón para avanzar una posición. # steps\step_reglas.py:76.

Then: se determina que el peatón en la posición 2,2, tiene (1,0) de movimiento en el camino para un paso. # steps\step_reglas.py:89.

4.6. Ejecución del escenario avanza un peatón dos celdas hacia adelante

A continuación, se presentan los resultados de la ejecución de la prueba en BDD de la Tabla 4 y los ejemplos definidos en la Figura 6, para el escenario donde se verifica el avance de dos celdas hacia adelante del peatón. Resultado de la ejecución de la prueba.

Tabla 4.
Plantilla BDD. Avanza un peatón dos celdas hacia adelante

Parte	Descripción
Scenario Outline	Avanza un peatón dos celdas hacia adelante, si puede.
Given (*Dado*)	Se definen las reglas del autómata celular.
When (*Cuando*)	Cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón <destino> y la posición <actual> del peatón para avanzar dos posiciones.
Then (*Entonces*)	Se determina que el peatón en la posición <actual>, tiene (<resultado>) de movimiento en el camino para dos pasos.

Fuente: elaboración propia.

Ejemplos: adelante_dos_movimiento

destino	actual	resultado
4,4	2,2	2,2
0,0	2,2	-2,-2
0,4	2,2	-2,2
4,0	2,2	2,-2
2,0	2,2	0,-2
2,5	2,2	0,2
0,2	2,2	-2,0
5,2	2,2	2,0

Figura 6. Ejemplos escenario avanza un peatón dos celdas hacia adelante
Fuente: elaboración propia.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.1 adelante_dos_movimiento # peatones.feature:83.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 4,4 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (2,2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.2 adelante_dos_movimiento # peatones.feature:84.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,0 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-2,-2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.3 adelante_dos_movimiento # peatones.feature:85.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,4 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-2,2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.4 adelante_dos_movimiento # peatones.feature:86.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 4,0 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (2,-2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.5 adelante_dos_movimiento # peatones.feature:87.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 2,0 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (0,-2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.6 adelante_dos_movimiento # peatones.feature:88.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 2,5 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (0,2) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.7 adelante_dos_movimiento # peatones.feature:89.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 0,2 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (-2,0) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

• **Scenario Outline**: avanza un peatón dos celdas hacia adelante, si puede. -- @1.8 adelante_dos_movimiento # peatones.feature:90.

• **Given**: se definen las reglas del autómata celular # steps\step_reglas.py:10.

• **When**: cuando se quiera realizar un movimiento de un peatón, se debe verificar si su movimiento es válido. Como ejemplo, dado un destino del peatón 5,2 y la posición 2,2 del peatón para avanzar dos posiciones. # steps\step_reglas.py:95.

• **Then**: se determina que el peatón en la posición 2,2, tiene (2,0) de movimiento en el camino para dos pasos. # steps\step_reglas.py:108.

4.7. Encuesta de la aceptación del uso de la metodología

Como era un primer ejercicio del uso de BDD en este tipo de proyectos, se propuso construir una encuesta sencilla como instrumento de retroalimentación en el proceso de desarrollo usando BDD (ver Tabla 5), para la cual se obtuvieron los siguientes resultados:

Tabla 5.
Encuesta de aceptación de la metodología

Preguntas	Muy en desacuerdo	En desacuerdo	No estoy seguro	De acuerdo	Muy de acuerdo
¿Siente usted que se cumplió con los tiempos propuestos en la planeación del proyecto?	0 %	0 %	0 %	40 %	60 %
¿El modelo construido en el proyecto tuvo en cuenta todos los elementos que usted considera?	0 %	0 %	0 %	20 %	80 %
¿El modelo funciona como usted espera?	0 %	0 %	0 %	30 %	70 %
¿Está usted satisfecho con el rendimiento del proceso de la construcción del modelo?	0 %	0 %	0 %	40 %	60 %
¿En su opinión, usted obtuvo retroalimentación todo el tiempo de lo que se estaba realizando en el proyecto?	0 %	0 %	0 %	0 %	100 %
¿El prototipo se podía ajustar durante el proyecto?	0 %	0 %	0 %	40 %	60 %
¿La calidad del producto es como usted lo esperaba?	0 %	0 %	0 %	30 %	70 %
¿Recomendaría usted el uso de la metodología BDD a otras personas?	0 %	0 %	0 %	0 %	100 %
¿Volvería a usar BDD como metodología para el desarrollo de un futuro proyecto?	0 %	0 %	0 %	0 %	100 %
¿Usted siente que hubo elementos que faltaron en la construcción del modelo?	100 %	0 %	0 %	0 %	0 %
¿Usted siente que el proyecto se estancaba y no avanzaba en el tiempo de ejecución?	80 %	20 %	0 %	0 %	0 %

Fuente: elaboración propia.

4.8. Ejecución del modelo

Como resultado final se presenta una de las ejecuciones del autómata celular y solo se muestra una de las ejecuciones, porque el campus de la Universidad del Valle es muy grande, siendo este el objeto de estudio para el cual se desarrolló el modelo (Vargas-Forero *et al.*, 2019). Se decidió mostrar la vía peatonal que conduce a la cafetería central, ya que está en la hora pico (al mediodía) y es la más concurrida en toda la universidad, mostrando el mayor número de peatones y congestión en la vía.

En la Figura 7 se observa la vía peatonal de la cafetería central, para esta ejecución hay 1.121 peatones que proceden de muchos orígenes, pero todos convergen al destino de la cafetería central. El tiempo de ejecución de esta imagen son 5000 ciclos, equivalente a 83,33 minutos dado que cada ciclo se asumió como un segundo en el tiempo. Se observa que los peatones fluyen sin contratiempo y de la misma forma fluyen los peatones en el resto de las vías peatonales.

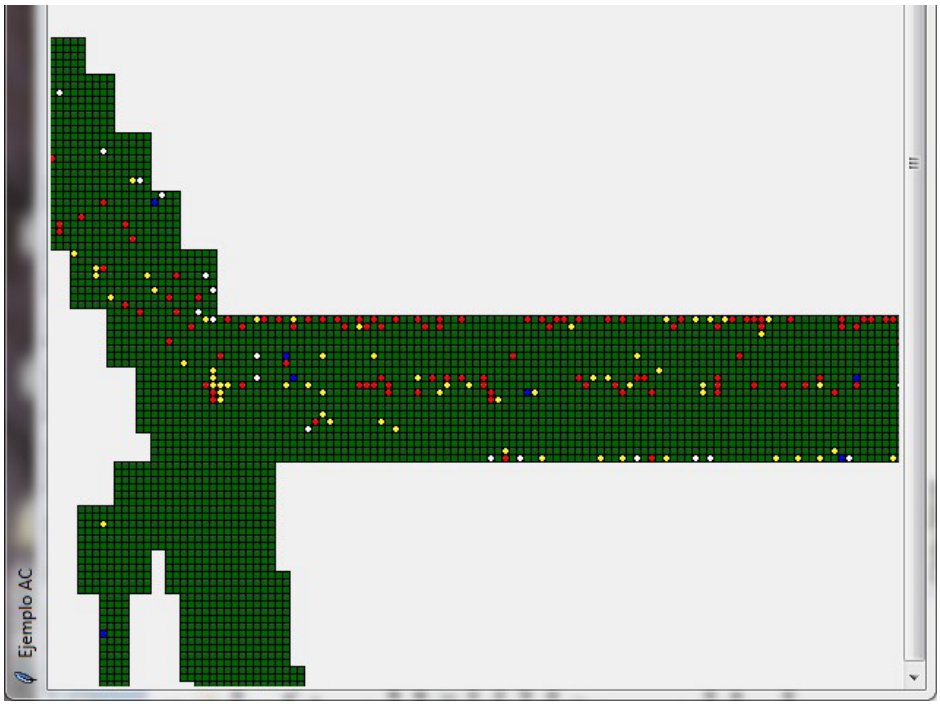

Figura 7. Ejecución del autómata celular con 1121 peatones y 5000 ciclos
Fuente: elaboración propia.

5. Conclusiones

Al momento del desarrollo del proyecto no se evidenció en la literatura el uso de BDD en aplicaciones de simulación de movimiento peatonal y menos con el uso de autómatas celulares, por tanto, se considera oportuna la construcción del primer modelo de autómata celular para una simulación de movimiento peatonal con el uso de BDD.

Al ser la primera implementación conocida con el uso de BDD una de las mayores expectativas alrededor de ella, los tiempos esperados o las características solicitadas no se cumplieron, sin embargo, al finalizar el proyecto se evidenció que dichos tiempos fueron adecuados y suficientes para los usuarios que solicitaron el software. El análisis y la especificación de los requisitos del proyecto se desarrollaron de forma natural para las personas que participaron en el proyecto, considerando que no contaban con formación en informática o sistemas. Gracias a la realización de pruebas continuas, el control de calidad del producto se pudo realizar de manera exitosa, rápida y eficiente.

En las etapas de la construcción, donde se encontró la necesidad de realizar cambios por nuevas características o modificaciones en el proyecto, la implementación e integración en el proceso de desarrollo se facilitó gracias a que dichas modificaciones no tuvieron impacto desfavorable en el producto, no hubo afectación en los tiempos ni tampoco en la calidad del producto. Como se puede evidenciar en la encuesta de aceptación acerca del uso de la metodología, los usuarios que participaron en el desarrollo mostraron una elevada aceptación en el uso la misma, dado que, por ejemplo, ninguna pregunta generó ambigüedad o incertidumbre para la respuesta del usuario; así mismo, la totalidad de respuestas se encontraron en los rangos de mayor aceptación (De acuerdo y Muy de acuerdo ó en Desacuerdo y Muy en desacuerdo), según la orientación de la pregunta.

La aplicación de BDD en proyectos de investigación podría resultar ser amplia y variada, sobre todo con implementaciones formales. Un ejemplo de futuras aplicaciones lo puede constituir Evento-B, donde se pueda realizar la transición de un documento de requisitos a una especificación formal.

6. Referencias

Beck, Kent; Beedle, Mike; Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin;… Thomas, Dave (2001). *Agilemanifesto.* Manifiesto por el Desarrollo Ágil de Software. Recuperado de https://agilemanifesto.org/iso/es/manifesto.html

Buitrago, Pablo; Kattán, José (2011). *Universidad del Valle arquitectura para la educación*. Cali, Colombia: Universidad del Valle.

Chen, Bi; Wang, Yafei; Wang, Donggen; Li, Qingquan; Lam, William; Shaw, Shih (2018). Understanding the impacts ofhuman mobility on accessibility usingmassivemobile phone tracking data. *Annals of the American Association of Geographers*, *108*(4), 1115–1133. https://doi.org/10.1080/24694452.2017.1411244

Engel, Jens; Rice, Benno; Jones, Richard (2012). *Behave.* Recuperado de https://behave.readthedocs.io/en/latest/

Ferguson-Smart, John (2014). *BDD in Action: Behavior-Driven Development for the whole software lifecycle*. Manning Publications.

Gipps, P. G.; Marksjö, B. (1985). A micro-simulation model for pedestrian flows. *Mathematics and Computers in Simulation*, *27*(2–3), 95–105. https://doi.org/10.1016/0378-4754(85)90027-8

Gudowski, Bartlomiej; Waș, Jaroslaw (28-30 de junio de 2007). Some criteria of making decisions in pedestrian evacuation algorithms. En *6th International Conference on Computer Information Systems and Industrial Management Applications, CISIM 2007* (pp. 93–96), Minneapolis, MN, USA. https://doi.org/10.1109/CISIM.2007.61

Jurado, Carlos (2010). *Diseño Ágil con TDD*. Lulu.com.

Khamis, Abdelaziz; Abdelmonem, Ashraf (2002). The unified software development process and framework. *Dogus University Dergisi 5,* 109-122. https://doi.org/10.31671/dogus.2019.348

Martins-Gonçalves, Natalia; Nuñez-Basantes, Alba (2017). Assessment the pedestrian accessibility in the brt stations in two cities of Latin America (breakout presentation). *Journal of Transport & Health*, *7*(Suplement), S76–S77. https://doi.org/10.1016/j.jth.2017.11.124

Muñoz Ceballos, L. E. (2017). *Herramienta para modelar la accesibilidad peatonal al interior del campus de la Universidad del Valle con autómatas celulares sobre una plataforma SIG.* Universidad del Valle, Cali, Colombia.

North, D. (2006). Introducing {BDD}. Retrieved January 12, 2019, from Better Software Magazine website: https://dannorth.net/introducing-bdd/

Poppendieck, Mary; Cusumano, Michael (2012). Lean Software Development: A Tutorial. *IEEE Software*, *29*(5), 26–32. https://doi.org/10.1109/MS.2012.107

Qin, Han; Curtin, Kevin; Rice, Matthew (2018). Pedestrian network repair with spatial optimization models and geocrowdsourced data. *GeoJournal*, *83*(2), 347–364. https://doi.org/10.1007/s10708-017-9775-x

Schadschneider, A. (2001). *Cellular Automaton Approach to Pedestrian Dynamics - Theory* (p. 75). p. 75. Springer 2001.

Solis, Carlos; Wang, Xiaofeng (2011). A Study of the Characteristics of Behaviour Driven Development. En *37th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 383–387), Oulu, Finland. https://doi.org/10.1109/SEAA.2011.76

Vargas-Forero,Victor; Muñoz-Ceballos, Luz; García-Baños, Ángel; Jaramillo-Molina, Ciro (2019). Modelo con autómatas celulares para analizar la accesibilidad peatonal al interior del campus universitario meléndez de la Universidad del Valle. *Scientia et Technica*, *24*(1), 67–75.

Was, J. (8-10 septiembre de 2005). Cellular automata model of pedestrian dynamics for normal and evacuation conditions. En *5th International Conference on Intelligent Systems Design and Applications 2005, ISDA '05* (154–159), Warsaw, Poland. https://doi.org/10.1109/ISDA.2005.31

Wolfram, M. (2019). *Moore Neighborhood*. Retrieved October 10, 1BC, from MathWorld Team website: http://mathworld.wolfram.com/MooreNeighborhood.html.