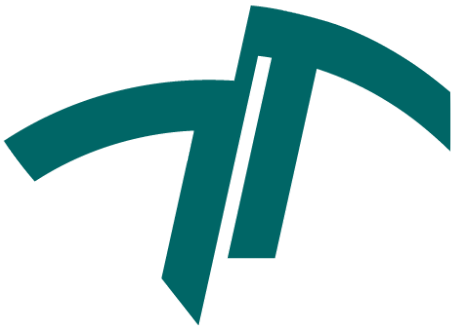


Taller de programacion 1
Trabajo Práctico

Alumnos:

Agustin Perea
Dario Funes
Kevin Zamora
Leonardo Motylicki

Grupo 6

Fecha de entrega: 12/11/2023

Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Índice

Introducción

Este informe detalla los resultados de las pruebas

- Caja negra
- Caja blanca
- Test de integración
- Test de persistencia
- Test de GUI

El test de caja negra busca verificar la funcionalidad de un sistema sin conocer los detalles internos de su implementación, es decir, que evalúa el comportamiento externo del software sin tener acceso al código fuente
En este trabajo se testean las clases :

en modelo de datos: Admin, Contratación,Ticket,Empleador,EmpleadoPretenso
en modelo de negocio: Agencia

El test de caja blanca también conocido como prueba estructural o de código, se centra en evaluar la lógica interna, la estructura y el código fuente de un programa. A diferencia del test de caja negra, este test requiere un conocimiento detallado de la implementación interna del software
En este trabajo se testea el método aplicaPromo() de la clase utilPromo

El test de integración es la fase en la que se combinan y prueban los diferentes módulos o componentes del sistema para asegurar que funcionan juntos de manera correcta y coherente. Busca detectar posibles problemas de interacción entre las partes del sistema y garantizar que la integración de esos componentes cumpla con los requisitos establecidos.
En este trabajo el test de integración abarca las secuencia de registro y login, gestión de ticket, elegir candidato, ronda encuentro, ronda contratación

El test de persistencia se centra en verificar la capacidad del sistema para almacenar y recuperar datos de manera correcta y confiable a lo largo del tiempo, es decir, que este tipo

de prueba se enfoca en asegurar que los datos persistan de manera duradera en el sistema, incluso después de reinicios, actualizaciones o fallos

El test de GUI se centra en evaluar la usabilidad, la apariencia y la funcionalidad de la interfaz visual del sistema, este tipo de prueba se realiza para garantizar que la interfaz de usuario cumpla con los requisitos de diseño y usabilidad.
En este trabajo el test de GUI abarca : GUILogin, GUICliente, GUIAdmin, GUIRegistro

Tests caja negra

El Test de Caja negra se realizó teniendo en cuenta los requerimientos y narrativa de las precondiciones e invariantes del jdoc del Sistema.

OVERVIEW PACKAGE CLASS USE TREE INDEX HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH: Search

Package modeloDatos

package modeloDatos

Classes

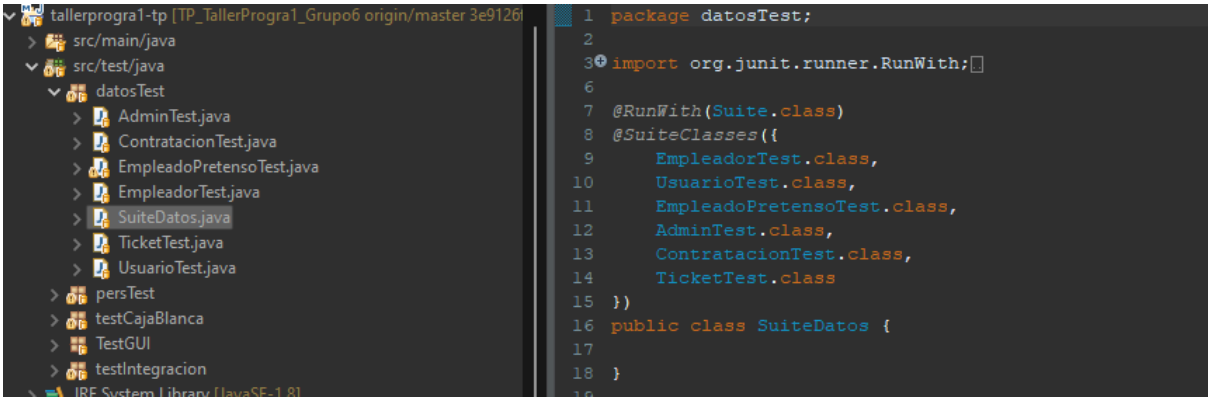
Class	Description
Admin	Clase que representa al Administrador del sistema.
Cliente	Clase abstracta que representa los Clientes, los cuales podran ser: Empleados Pretensos o Empleadores. Los atributos ticket y listaDePostulantes pueden ser null El atributo **puntaje** puede tomar cualquier valor numerico (tanto positivo, cero, o negativo) **Todos los setters tienen como precondicion , que su parametro es diferente de null**
ClientePuntaje	Clase que representa un Cliente con su corespondiente puntaje obtenido en una busqueda laboral El objeto es Comparable en forma descendente por su puntaje obtenido.
Contratacion	Clase que representa una contratacion entre un empleado y un empleador con su correspondiente fecha de creacion **Todos los setters tienen como precondicion , que su parametro es diferente de null Invariante de clase,** **Los atributos empleado, empleador y fecha son diferentes de null**
EmpleadoPretenso	Clase que representa un usuario pretenso **Todos los setters tienen como precondicion , que su parametro es diferente de null** **Invariante de clase** El atributo apellido es diferente de null El atributo edad es siempre positivo.
Empleador	Clase que representa un Usuario de tipo Empleador. **Todos los setters tienen como precondicion, que su parametro es diferente de null.**
Ticket **Clase que representa un ticket de empleo pretendido** **Todos los setters de tipo String tienen como precondicion , que su parametro es diferente de null y son del tipo esperado contemplado en la clase Constantes** Invariante de clase: **Todos los atributos de tipo String son diferentes de null y son del tipo esperado contemplado en la clase Constantes** **Remuneracion es mayor que cero.**	
Usuario **Clase abstracta que representa los Usuarios.**	

Se generaron las tablas de particiones y las baterías de pruebas de las clases perteneciente a los paquetes modeloDatos y modeloNegocio y persistencia.

Las Particiones de equivalencia de las clases están detalladas en el archivo excel dentro del repositorio con nombre:

TPTallerdeProgramacion1.xlsx

Creación de Pruebas:
Las baterías de pruebas fueron codificadas y ejecutadas utilizando JUnit 4, facilitando la automatización y repetición de las pruebas.



Ejecución de Pruebas:

Las pruebas fueron ejecutadas utilizando los conjuntos de datos diseñados, observando el comportamiento del sistema y registrando cualquier desviación del comportamiento esperado.

Errores encontrados:

1) La clase Ticket
   En el paquete datos se encuentra la clase Ticket. Al hacer las pruebas, las mismas arrojan fallas en 3 métodos distintos. El análisis y descripción de estos pruebas se detallará en las lineas que siguen.
   La clase ticket cuenta con diferentes métodos para obtener un puntaje enfrentando un ticket con otro. El puntaje se obtiene mediante una tabla por cada atributo donde se asigna Existe entonces un método para cada una de las 6 comparaciones que se hace y uno más para la comparación total.
   En la batería de pruebas se encontraron fallas en el método getComparacionPuesto y getComparacionJornada. En ambos casos la prueba que detecta es la falla es la n° 9 que es cuando los atributos a comparar son iguales (jornada extendida y puesto management) y el puntaje debió ser 1.
   Se deja asentado que, de haber fallado alguna de las pruebas anteriores, los métodos testGetComparacionPuesto y testGetComparacionJornada deberían haberse desdoblado en tests diferentes para hacer saltar o no cada assert de manera individual en cada prueba.
   Finalmente en el testGetComparacionTotal se plantearon dos escenarios, uno en el cual todos los atributos son iguales y otro en el que son todos distintos, fallando el primero puesto que se construyeron tickets con atributo jornada extendida.
2) La clase EmpleadoPretenso
   En esta clase tras realizar las pruebas 5 y 6 del método double calculaComision(Ticket ticket) se encontraron fallas.
   En el caso de la prueba 5 se pasó por parámetro un ticket inicializado con remuneración 1000 y puesto SENIOR para obtener una comisión de 900.0 pero se obtuvo 1050. En el segundo caso el ticker se inicializó con remuneración 1000.0 y puesto MANAGMENT para obtener una comisión deo 500, pero se obtuvo una de 499.99.
3) La clase Empleador

En este caso se detectaron fallas en el constructor. Concretamente en el parámetro del nombre y del teléfono. Por lo que surge de las pruebas unitarias, ambos campos se cargan al revés.

4) La clase Agencia
    a) El método testCalculaPremiosCastigosAsignaciones2 de la clase CalculaPremiosCastigosAsignacionesTest arroja una falla cuando se realiza una prueba con datos válidos en la que un empleado debería perder 5 puntos -salida esperada-, pero no pierde nada.
    b) La clase CrearTicketEmpleadoTest detecta una falla en el método crearTicketEmpleado lanzando una excepción cuando no debería. Esto ocurre al testear el escenario en el que se prueba la agencia con un empleado con un ticket.
    c) En la clase CrearTicketEmpleadorTest también detecta una falla en el mismo escenario que en la precedente clase.
    d) En la clase TestMatch se observa una falla ya que no se calculan bien las comisiones del usuario en este caso empleador
    e) En la clase SetLimitesRemuneracionTest hay una falla en la prueba n°2 que es justamente la que no debería arrojar ninguna excepción ya que ambos valores pasados por paráemetro son válidos.
    f) En la clase RegistroEmpleadorTest hay una falla en la prueba n° 1 del escenario 1 que es, al igual que el caso anterior, la que contiene todos valores válidos pasados por parámetro. La falla marcada corresponde al atributo nombre, misma falla que en el constructor del empleador.

Pruebas unitarias del paquete persistencia

El paquete persistencia guarda las clases PersistenciaXML –que se extiende de la interfaz IPersistencia-, AgenciaDTO y UtilPersistencia. Los constructores y métodos fueron testeados bajo la metodología de caja negra y con los test unitarios que se describen en las líneas que siguen.

Para la clase PersistenciaXML se proyectó el escenario de que el sistema se persiste ya sea a pedido del usuario o por default al cerrar sesión. Si bien no existen restricciones en el contrato, dada las características del sistema y concretamente la persistencia, no se prevén otro tipo de escenarios. O crea la persistencia o no la crea. La prueba de este escenario se ejecutará en la clase Agencia.

Con respecto a los métodos, conforme a la interpretación de los mismos, se plantearon los diferentes escenarios a testear en los diferentes métodos:

1) public void testAbrirInput() throws IOException {…}

Utilizado para testear el método void abrirInput(String nombre). Se plantea un escenario 1, en el cual se recibe un parámetro String con el nombre del archivo con una extensión válida en el que está guardada la persistencia. La salida esperada es que el archivo se abra. Si bien no está en el contrato, no se puede predecir qué ocurriría si el archivo no existe o si se ingresa por parámetro un tipo de archivo con extensión errónea, por lo tanto no se lo considera un escenario a testear. Se plantea entonces un escenario 2, que se trata de que

el método reciba un un parámetro null. En dicho caso la salida esperada es una excepción de tipo null pointer.

Al ejecutar las pruebas con junit 4 efectivamente ambas salidas son las esperadas.

2) public void testAbrirOutput() throws IOException {…}

Este método testea el método void abrirOutput(String nombre). Se interpreta que la intención es abrir el archivo para guardar los datos mediante una capa de persistencia. Al igual que anteriormente, se plantean los escenarios predecibles que son que pase un parámetro correcto y un null. Nuevamente al igual que el caso anterior la salida coincidió con la esperada.

3) public void TestCerrarInput() {…}

Este método se utilizó para testear el void cerrarInput (). Se interpretó que este método se encarga de cerrar el archivo abierto en void abrirInput(String nombre) puesto que no requiere ningún parámetro.

4) public void TestCerrarOutput() {…}

Análogamente con el apartado anterior, primero se ejecutó el método abrirOutput pasando por parámetro el nombre del archivo “persistencia.xml”.

5) public void testLeer() throws ClassNotFoundException {…}

En este caso se interpreta que el método lee el archivo y extrae de él la información guardada. El escenario planteado es que se ejecuta correctamente y efectivamente así ha sido el resultado. Particularmente este método devuelve un objeto genérico, por lo que, de manera coherente con la interpretación, se utiliza una variable de tipo AgenciaDTO.

6) public void testEscribir1 (){…} y public void testEscribir2 (){…}

Este método testea el método void escribir (Object objeto). Al igual que en el caso anterior, se interpreta que el objeto genérico se trata de una AgenciaDTO. Se plantean dos escenarios, uno en el que se recibe un parámetro vacío y otro con información. En ambos casos el método se ejecuto correctamente guardando una AgenciaDTO vacía en el archivo archivo1.xml y una con datos válidos en el archivo archivo2.xml

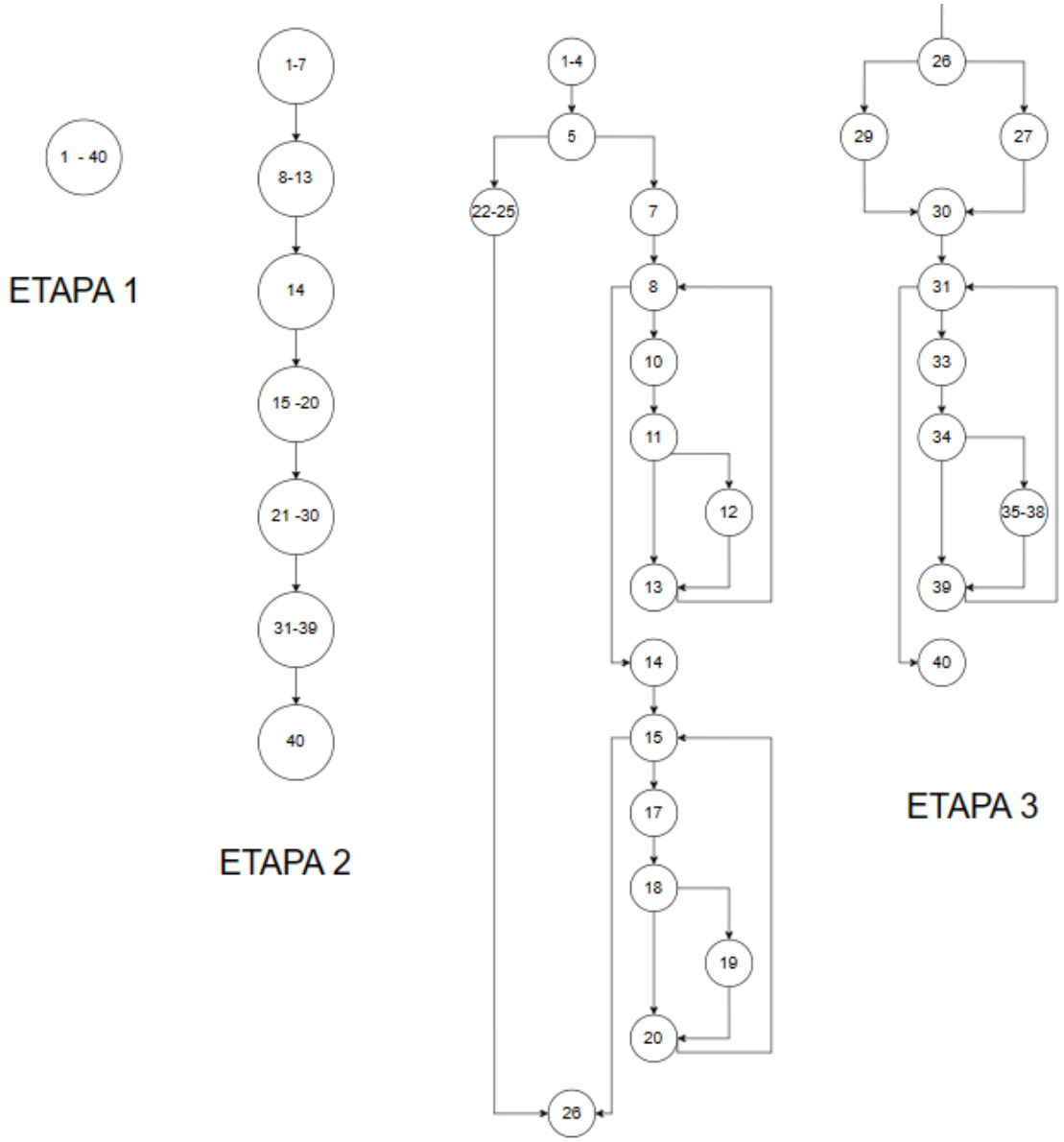
La clase AgenciaDTO tiene un constructor vacío efectivamente para utilizar el patrón adecuado para la persistencia en un archivo xml. Es por eso que la forma de testear esta clase fue instanciar sus atributos, asignarles valores válidos y utilizar los setters de la clase para comprobar que esta clase es efectivamente funcional. La salida esperada fue la obtenida.

Con respecto a la clase UtilPersistencia, se trata de la clase que “traduce” los elementos a persistir de un objeto Agencia a uno AgenciaDTO y viceversa. El objetivo de estas pruebas es corroborar que puede copiar correctamente los elementos de una clase y pasarlo a otra. Como Agencia es una clase que no se puede instanciar nuevamente, se utilizará el método

Agencia.getInstance(). En ambos métodos las salidas esperadas eran que los objetos copiados sean idénticos a los originales y efectivamente así fue.

Tests caja blanca

Construcción del grafo

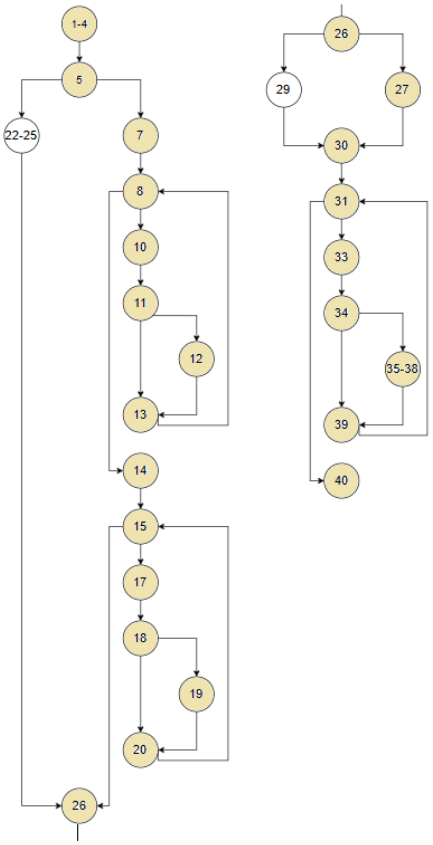


calculo complejidad ciclomática

numero de arcos + 2 - número de nodos /-> 33 + 2 - 26 = 9
verificó con otra forma de calcularlo , la cual es: numero de ciclos +1 -> = 9
La construcción del grafo se realizó por el método descendente y los caminos se eligieron por el método general

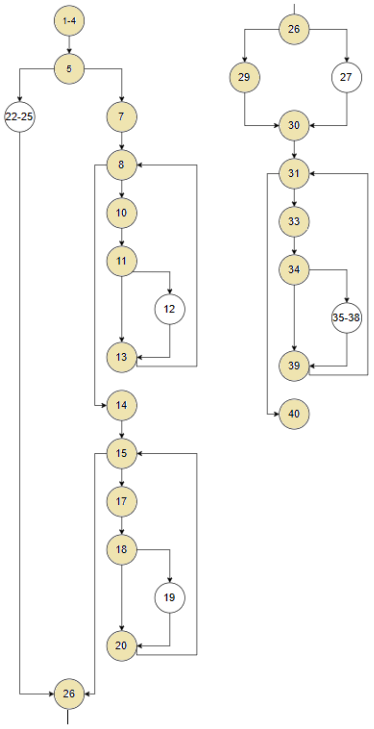
camino 1

1-4 / 5 / 7 / 8 / 10 / 11 / 12 / 13 / 8 / 14 / 15 / 17 / 18 / 19 / 20 / 15 / 26 / 27 / 30 / 31 / 33 / 34 / 35-38 / 39 / 31 / 40



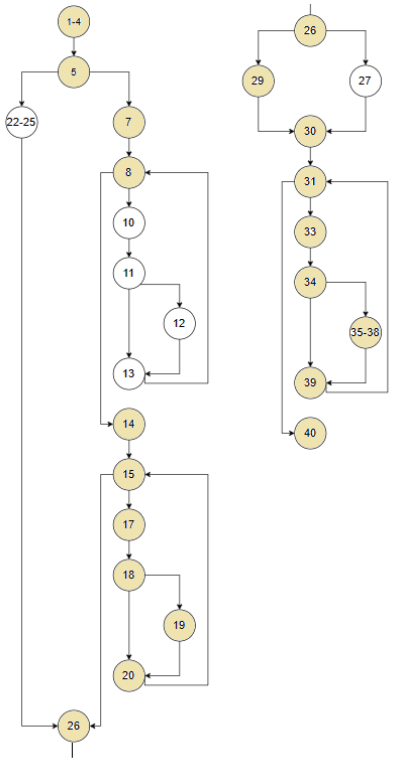
camino 2

1-4 / 5 / 7 / 8 / 10 / 11 / 13 / 8 / 14 / 15 / 17 / 18/ 20 / 15 / 26 / 29 / 30 / 31 / 33 / 34/ 39 / 31 / 40



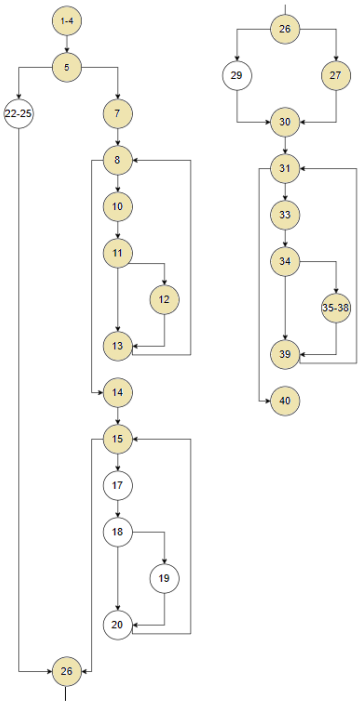
camino 3

1-4 / 5 / 7 / 8 / 14 / 15 / 17 / 18 / 19 / 20 / 15 / 26 / 29 / 30 / 31 / 33 / 34 / 35-38 / 39 / 31 / 40



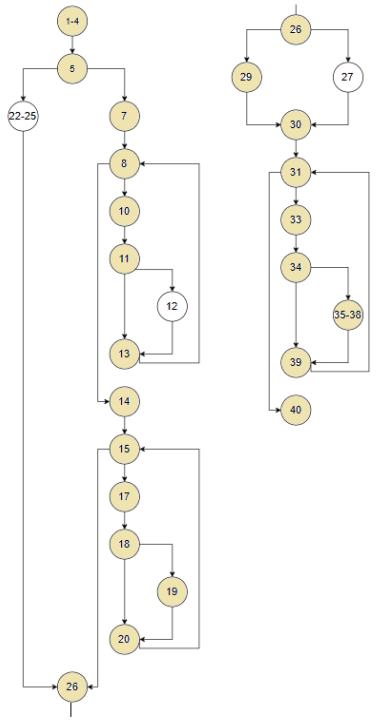
camino 4

1-4 / 5 / 7 / 8 / 10 / 11 / 12 / 13 / 8 / 14 / 15 / 26 / 27 / 30 / 31 / 33 / 34 / 35-38 / 39 / 31 / 40



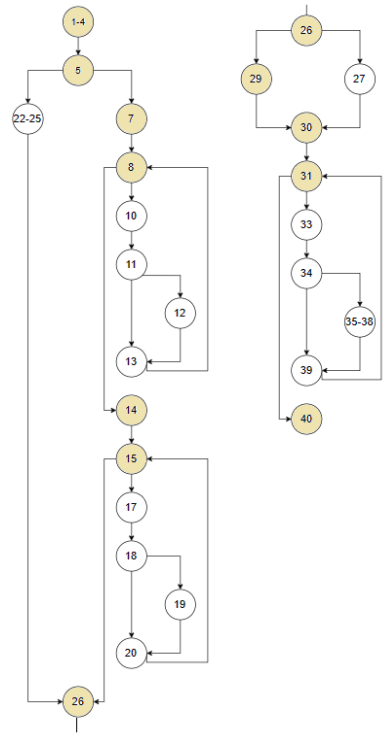
camino 5

1-4 / 5 / 7 / 8 / 10 / 11/ 13 / 8 / 14 / 15 / 17 / 18 / 19 / 20 / 15 / 26 / 29 / 30 / 31 / 33 / 34 / 35-38 / 39 / 31 / 40



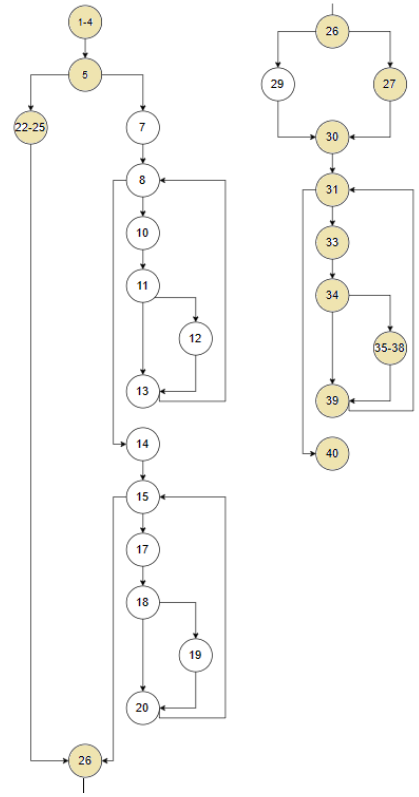
camino 6

1-4 / 5 / 7 / 8 / 14 / 15 / 26 / 29 / 30 / 31 / 40



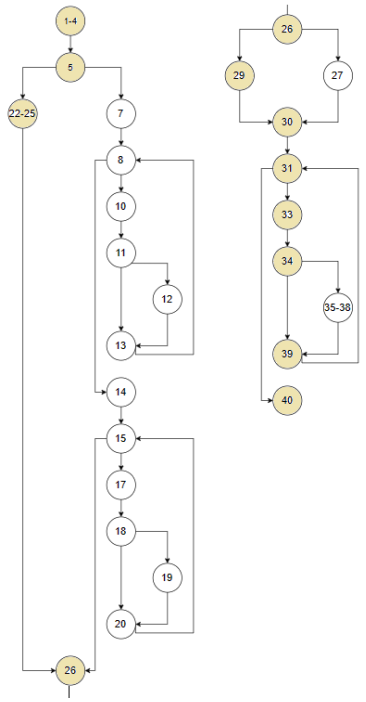
camino 7

1-4 / 5 / 22-25 / 26 / 27 / 30 / 31 / 33 / 34 / 35-38 / 39 / 31 / 40



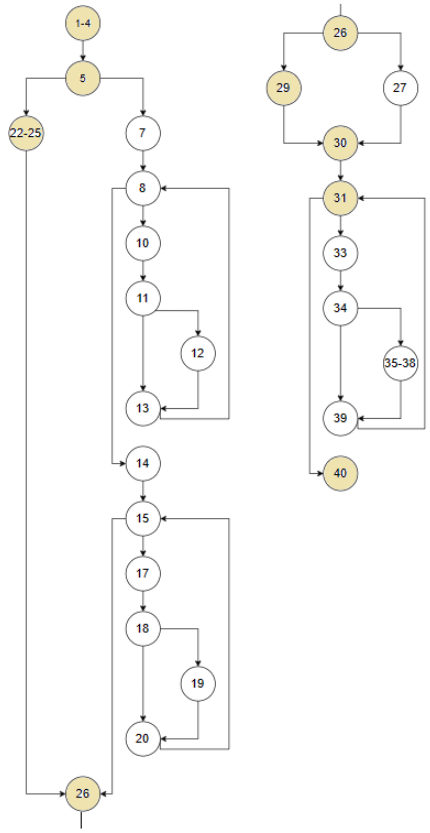
camino 8

1-4 / 5 / 22-25 / 26 / 29 / 30 / 31 / 33 / 34 / 39 / 31 / 40



camino 9

1-4 / 5 / 22-25 / 26 / 29 / 30 / 31 / 40



Cabecera del metodo

public Cliente aplicaPromo(
boolean promoPorListaDePostulantes,
HashMap<String, EmpleadoPretenso> empleados,
HashMap<String, Empleador> empleadores)

c°n	proceso	resultado esperado
1	promoPorListaDePostlantes = tue itEmpleadores.hasNext() = true empleador.getListaDePostulantes() != null itEmpleados.hasNext() =true empleadoPretenso.getListaDePostulantes() != null contadorEmpleador > contadorEmpleadoPretenso clientes.hasNext() = true cl.getPuntaje() > puntajeMaximo	empleado que tenga el puntaje más alto
2	promoPorListaDePostlantes = true itEmpleadores.hasNext() = true itEmpleados.hasNext() =true contadorEmpleador >contadorEmpleadoPretenso false clientes.hasNext() = true	no se espera nada, entonces null
3	promoPorListaDePostlantes = true itEmpleadores.hasNext() = FALSE itEmpleados.hasNext() =true empleadoPretenso.getListaDePostulantes() != null contadorEmpleador > contadorEmpleadoPretenso false clientes.hasNext() = true cl.getPuntaje() > puntajeMaximo	empleado que tenga el puntaje más alto
4	promoPorListaDePostlantes = true itEmpleadores.hasNext() = true empleador.getListaDePostulantes() != null itEmpleados.hasNext() =false contadorEmpleador > contadorEmpleadoPretenso	empleador que tenga el puntaje más alto

	clientes.hasNext() = true cl.getPuntaje() > puntajeMaximo	
5	promoPorListaDePostlantes = tue itEmpleadores.hasNext() = true itEmpleados.hasNext() =true empleadoPretenso.getListaDePostulantes() != null contadorEmpleador > contadorEmpleadoPretenso false clientes.hasNext() cl.getPuntaje() > puntajeMaximo	empleado que tenga el puntaje más alto
6	promoPorListaDePostlantes = true itEmpleadores.hasNext() = false itEmpleados.hasNext() = false contadorEmpleador > contadorEmpleadoPretenso clientes.hasNext() = false	no se espera nada, entonces null
7	promoPorListaDePostlantes = false contadorEmpleador > contadorEmpleadoPretenso clientes.hasNext() cl.getPuntaje() > puntajeMaximo true	empleador que tenga el puntaje más alto
8	promoPorListaDePostlantes = false contadorEmpleador > contadorEmpleadoPretenso = false clientes.hasNext()	no se espera nada, entonces null
9	promoPorListaDePostlantes = false contadorEmpleador > contadorEmpleadoPretenso = false clientes.hasNext() false	no se espera nada, entonces null

Las pruebas realizadas se manejaron en diferentes escenarios, siendo

Empleados vacio y Empleadores Vacios
Empleados vacio/no vacío y Empleadores no vacio/vacio
Empleados no vacio y Empleadores no vacio

Dado que la complejidad ciclomática nos dio un valor de 9 y basándonos en el trabajo de McCabe nos agrupamos en la categoría de métodos sencillos, sin muchos riesgos.
El resultado del test de cobertura no arrojó fallos con los caminos elegidos

Test GUI

Objetivo de las Pruebas
El objetivo principal de estas pruebas es evaluar la funcionalidad, usabilidad y rendimiento de las interfaces gráficas de usuario del Sistema, Se testean los módulos de Login, Registro, Panel de Administrador y Panel de Clientes.

Enfoque de Pruebas
Se adoptó un enfoque de pruebas automatizadas para garantizar la cobertura. Las pruebas se centraron en la validación de la interacción del usuario con los diferentes componentes de la interfaz.

Casos de Prueba

GUI Login	
public void testBotonRegistro()	Se clickea el boton registro, debe abrirse el panel de registro
public void testVacio()	Se comprueba que el boton login este activado
public void testSoloContrasena()	Se llena el campo de contraseña, Se comprueba que el boton login este activado
public void testSoloNombre()	Se llena el campo de nombre, Se comprueba que el boton login este activado
public void testAmbosLlenos()	Se llenan ambos, Se comprueba que el boton login este activado
public void testLoginUsuarioDesconocido()	Se ingresa un usuario desconocido, Se comprueba que el mensaje sea correcto
public void testAdminLog()	Se llenan los campos con el usuario admin, Se comprueba si se habre el panel Admin
public void testLoginPassErroneo()	Se ingresa un usuario con contraseña incorrecta, Se comprueba que el mensaje sea el correcto
testEstadoInicialEnCambioPanelRegistro()	Se entra a un panel de Registro y se regresa al panelLogin, Se comprueba que los campos nombredeUsuario y contraseña esten vacios por defecto
testEstadoInicialEnCambioPanelAdmin()	Se entra a un panel de Admin y se regresa al panelLogin, Se comprueba que los campos nombredeUsuario y contraseña esten vacios por defecto
GuiRegistro	
testBotonCancelar()	Se clickea el boton cancelar,
testRegVacio()	Se comprueba que el boton de registro este desactivado
testRegSoloNombre()	Se llena solo el campo de nombre, Se comprueba que el boton de registro este desactivado
testRegSoloContrasenia()	Se llena solo el campo de contraseña, Se comprueba que el boton de registro este desactivado

testRegSoloConfirmarContrasenia()	Se llena solo el campo de confirmarContraseña, Se comprueba que el boton de registro este desactivado
testRegSoloNombreReal()	Se llena solo el campo de nombreReal, Se comprueba que el boton de registro este desactivado
testRegSoloTelefono()	Se llena solo el campo de Telefono, Se comprueba que el boton de registro este desactivado
testRegSoloApellido()	Se llena solo el campo de Apellido, Se comprueba que el boton de registro este desactivado
testRegSoloEdad()	Se llena solo el campo de Edad, Se comprueba que el boton de registro este desactivado
testRegSinNombre()	Se llenan todos los campos excepto el de Nombre, Se comprueba que el boton de registro este desactivado
testRegSinContrasenia()	Se llenan todos los campos excepto el de Contraseña, Se comprueba que el boton de registro este desactivado
testRegSinConfirmarContrasenia()	Se llenan todos los campos excepto el de ConfirmarContraseña, Se comprueba que el boton de registro este desactivado
testRegSinNombreReal()	Se llenan todos los campos excepto el de NombreReal, Se comprueba que el boton de registro este desactivado
testRegSinTelefono()	Se llenan todos los campos excepto el de Telefono, Se comprueba que el boton de registro este desactivado
testRegSinApellido()	Se llenan todos los campos excepto el de Apellido, el checkbox empleador,Se comprueba que el boton de registro este desactivado
testRegSinEdad()	Se llenan todos los campos excepto el de Edad, el checkbox empleador, Se comprueba que el boton de registro este desactivado
testRegCorrecto()	Se llenan todos los campos, se comprueba que el boton de registro este activado
testReg1Empleador()	Se llenan todos los campos y el checkbox empleador, se clickea registro, Se comprueba que se creo un empleador y cambia de panel a PanelCliente
testReg1Empleado()	Se llenan todos los campos y el checkbox empleado, se clickea registro, Se comprueba que se creo un empleado y cambia de panel a PanelCliente
testRegPassNoCoincide()	Se llenan todos los campos con discrepancia en Contraseña y ConfirmarContraseña, se clickea registro, Se comprueba que se reciba el optionPanel con el mensaje correcto
testRegUsuarioRepetido()	Se llenan todos los campos pero el usuario ya existe en el sistema, se clickea registro, Se comprueba que se reciba el optionPanel con el mensaje correcto
testCambioDePanel()	Se comprueba que al crear un empleado se cambia de panel a PanelCliente
GUI Admin	
testModificarValoresSoloInferior()	Se llena solo el campo textoInferior, Se comprueba que el boton modificarValores este desactivado

testModificarValoresSoloSuperior()	Se llena solo el campo textoSuperior, Se comprueba que el boton modificarValores este desactivado
testModificarValoresInvalido()	Se llenan los campos, Se comprueba que el boton modificarValores este desactivado en los casos invalidos
testModificarValoresValido()	Se llenan los campos, Se comprueba que el boton modificarValores este activado, se clickea y se comprueba que se actualice el panel
testCerrarSesion()	Se clickea el boton CerrarSesion, Se comprueba que regrese al panelLogin
testGatillar()	Se clickea el boton Gatillar, Se comprueba que el titulo cambie al Mensaje correcto
testAplicarPromo()	Se clickea el boton AplicarPromo, se comprueba que no haga ningun cambio cuando no hay datos en la agencia
testVistaIsPorTicket()	Se clickea el checklist listaPostulantes, Se comprueba que el metodo de la vista coincida en valor con el del componente
testAplicarPromoEmpleado()	Se selecciona un empleado de la lista, se clickea el boton AplicarPromo, Se comprueba que salga una ventana emergente con un mensaje valido
testAplicarPromoEmpleadoSinListaPostulantes()	Se selecciona un empleado de la lista, se clickea el boton PorListaDePostulantes,se clickea el boton AplicarPromo, Se comprueba que salga una ventana emergente con un mensaje valido
testAplicarPromoEmpleador()	Se selecciona un empleador de la lista, se clickea el boton AplicarPromo, Se comprueba que salga una ventana emergente con un mensaje valido
testAplicarPromoEmpleadorSinListaPostulantes()	Se selecciona un empleador de la lista, se clickea el boton PorListaDePostulantes,se clickea el boton AplicarPromo, Se comprueba que salga una ventana emergente con un mensaje valido
GUI Clientes	
testCerrarSesion()	Se clickea el boton cerrar sesion, se comprueba que regrese al LoginPanel
testAreaDeTexto()	Se comprueba que el area de texto del ticket tenga el mensaje correcto
testSeleccionarCandidatoVacio()	Se clickea seleccionar candidato sin seleccionar ningun cliente, se comprueba que no suceda nada
testSeleccionarCandidato()	Se clickea seleccionar candidato con un cliente seleccionado en lista de candidatos, se comprueba que se abra un panel con el mensaje correcto
testCamposNuevoTicket()	Se clickea nuevo Ticket, se comprueban que los componentes actuen de la manera esperada
testCreacionNuevoTicket()	Se clickea nuevo Ticket, se llena el texfield remuneracionPretendida y se crea un nuevo ticket, se comprueba que los componentes actuen de la manera esperada
testEliminacionTicket()	Se clickea eliminarTicket, se comprueba que los componentes involucrados actuen de manera esperada

Resultado de las Pruebas
Se realizaron 63 pruebas en total en diferentes escenarios de las cuales 11 tuvieron fallos.

Problemas Detectados:

Panel Admin
Botón Aplicar promo:
-cuando la agencia esta vacia tocar el boton lanza NullPointerException(2 Test fallidos)
-cuando se les aplican la promo a un empleado o empleador no sale una ventana emergente con el mensaje correspondiente(4 Tests fallidos)

Panel Cliente
Área de Texto en Ticket:
-al no tener ningún ticket se muestra un mensaje incorrecto(1 Test fallido)

Panel Login
Log Usuario desconocido:
-Al loguearse con un usuario desconocido la ventana emergente no muestra un mensaje correcto (1 Test fallido)

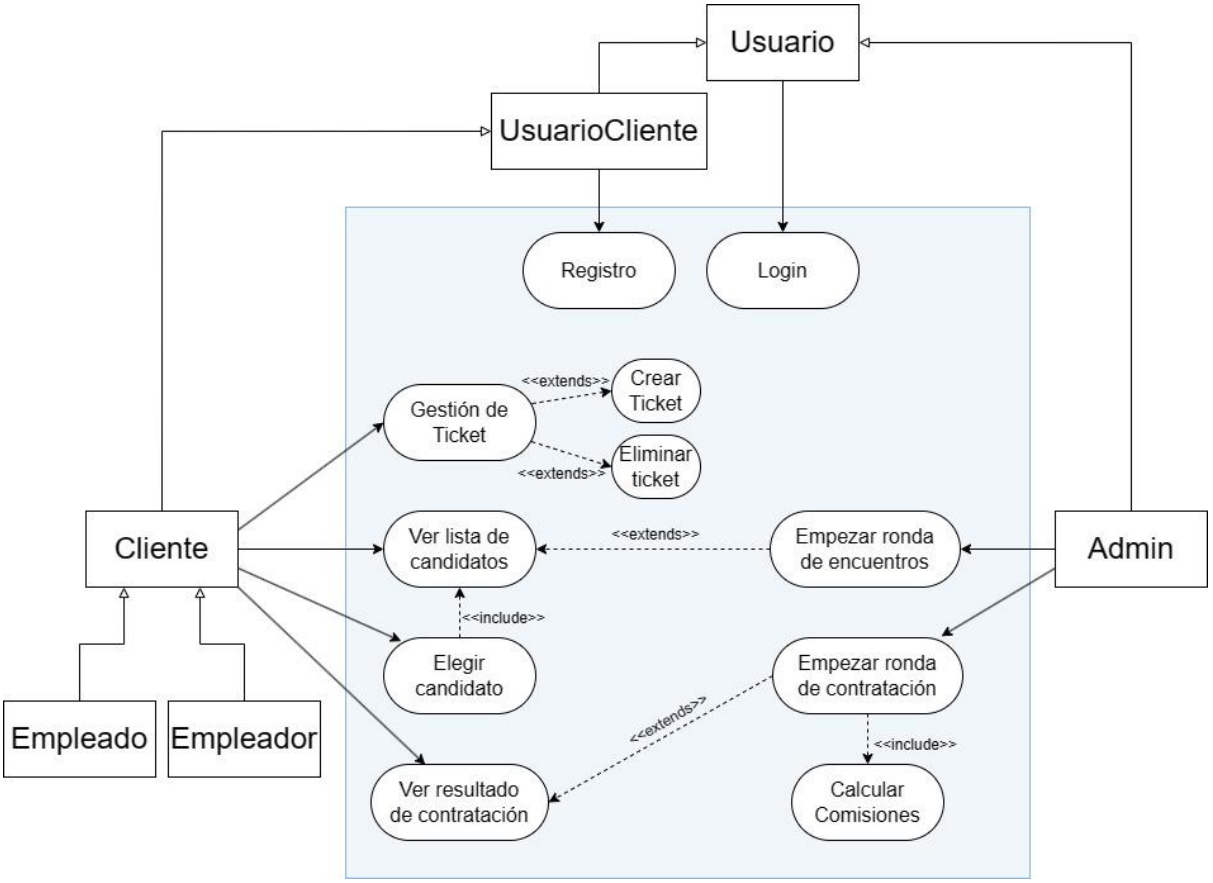
Cambio de Panel:
-al cerrar sesion de cualquier otro panel y regresar al Panel Login, no se resetean los campos nombre de usuario y contraseña a los valores por defecto(2 Test fallidos)

Panel Registro
Pass No coincide:
-Al registrar un usuario cuya contraseña no coincide con el campo confirmarContraseña, el sistema lo registra igualmente(1 Test Fallido)

Conclusion
En el proceso de prueba identificó algunas áreas en la GUI que requieren atención y corrección.

Tests de Integración

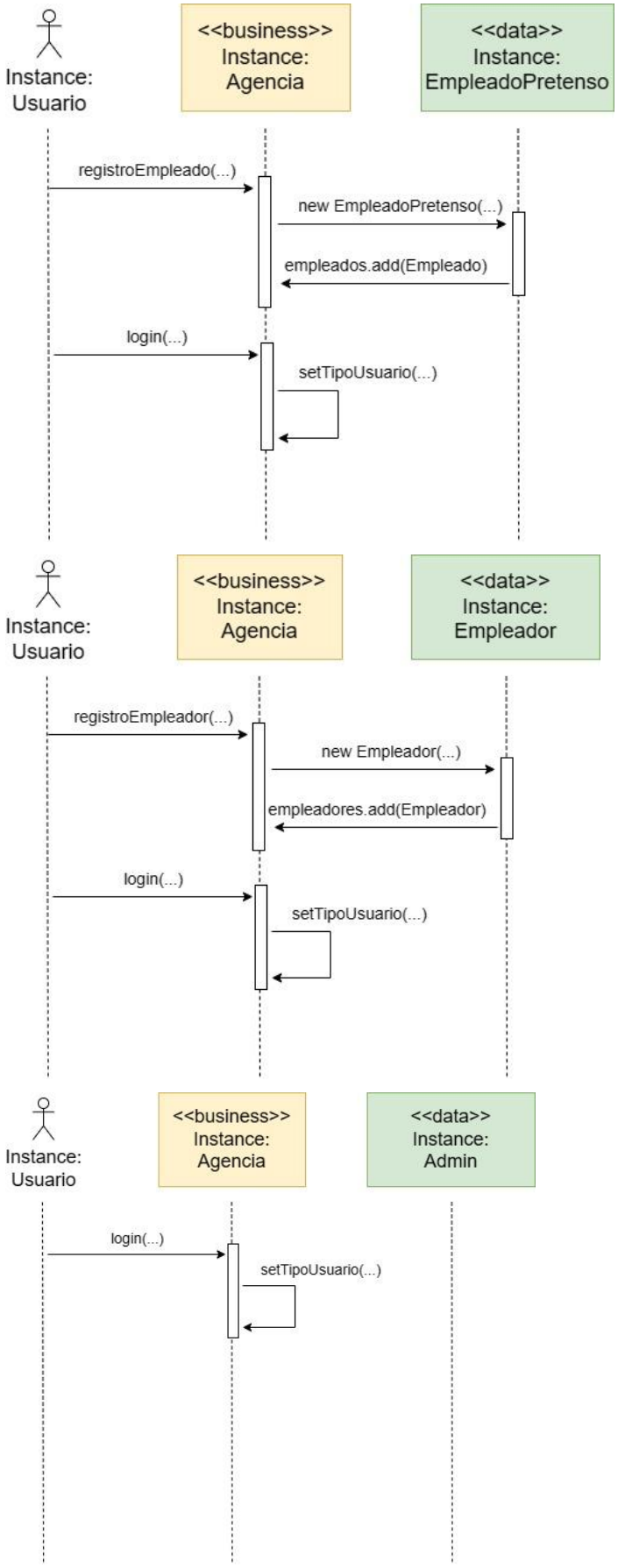
Casos de uso:



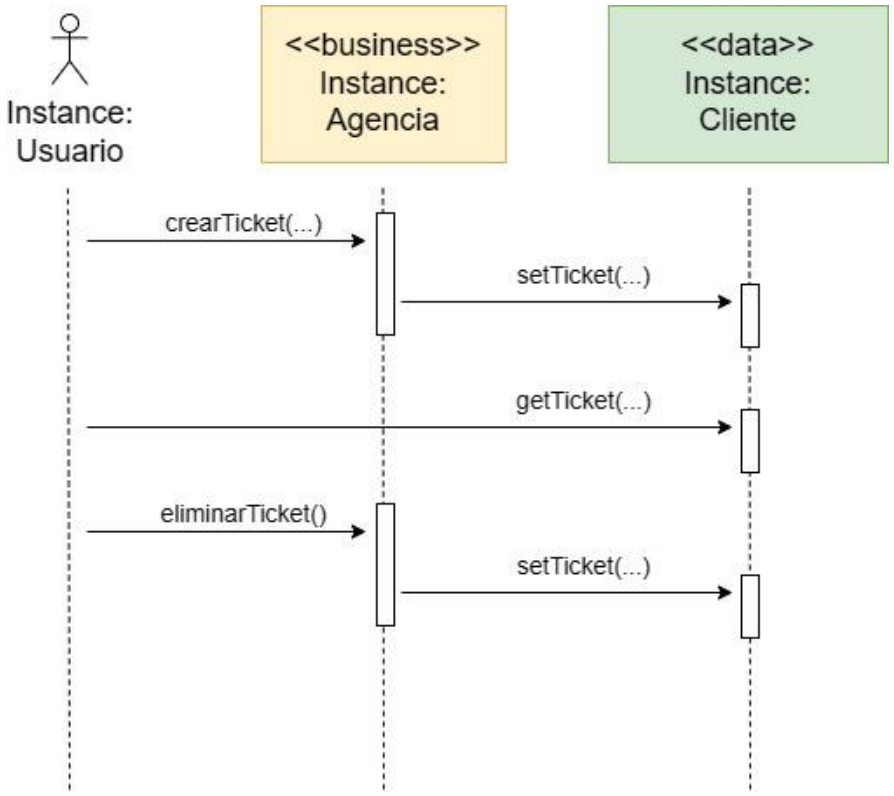
1. **Registro** es conveniente hacerlo con Empleado y Empleador, ya que cada uno necesita datos distintos para registrarse. Para el **Login** también debemos incluir al Admin.
2. La **Gestión de Ticket** es similar para ambos tipos de Cliente.
3. **Ver lista de candidatos** se testeara junto a **Elegir candidato**, ya que lo incluye y de otro modo sería redundante su testeo.
4. **Ver resultado de contratación** resulta un caso de uso simple y además depende en cierta medida de **Empezar ronda de contratación** por lo que se NO lo testeara por sí solo, si no en conjunto más adelante.
5. Testear **Empezar ronda de encuentros** junto a **Ver lista de candidatos**.
6. Testear **Empezar ronda de contratación** junto a **Ver resultado de contratación**.

Importante: Por fuera de los casos de uso de Registro y Login, en el resto queda implícito que el usuario está logueado, ya sea Cliente o Admin. Por lo que no se incluirá en los próximos diagramas.

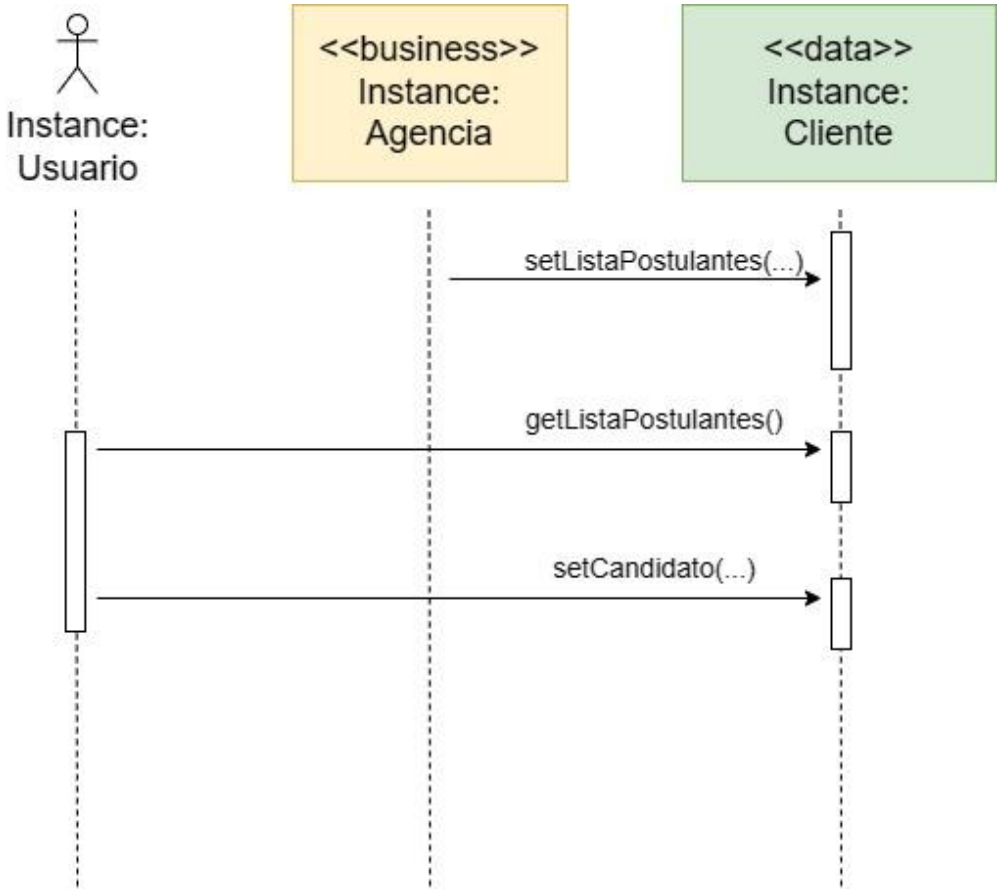
Secuencia 1: Registro y Login



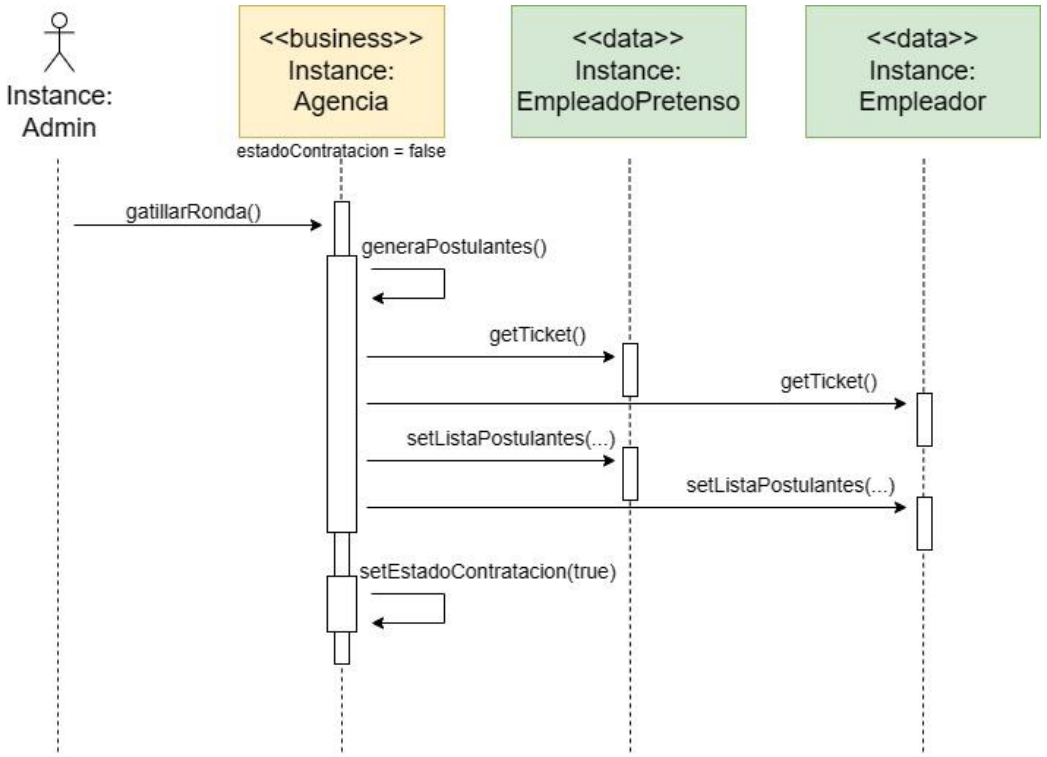
Secuencia 2: Gestión Ticket



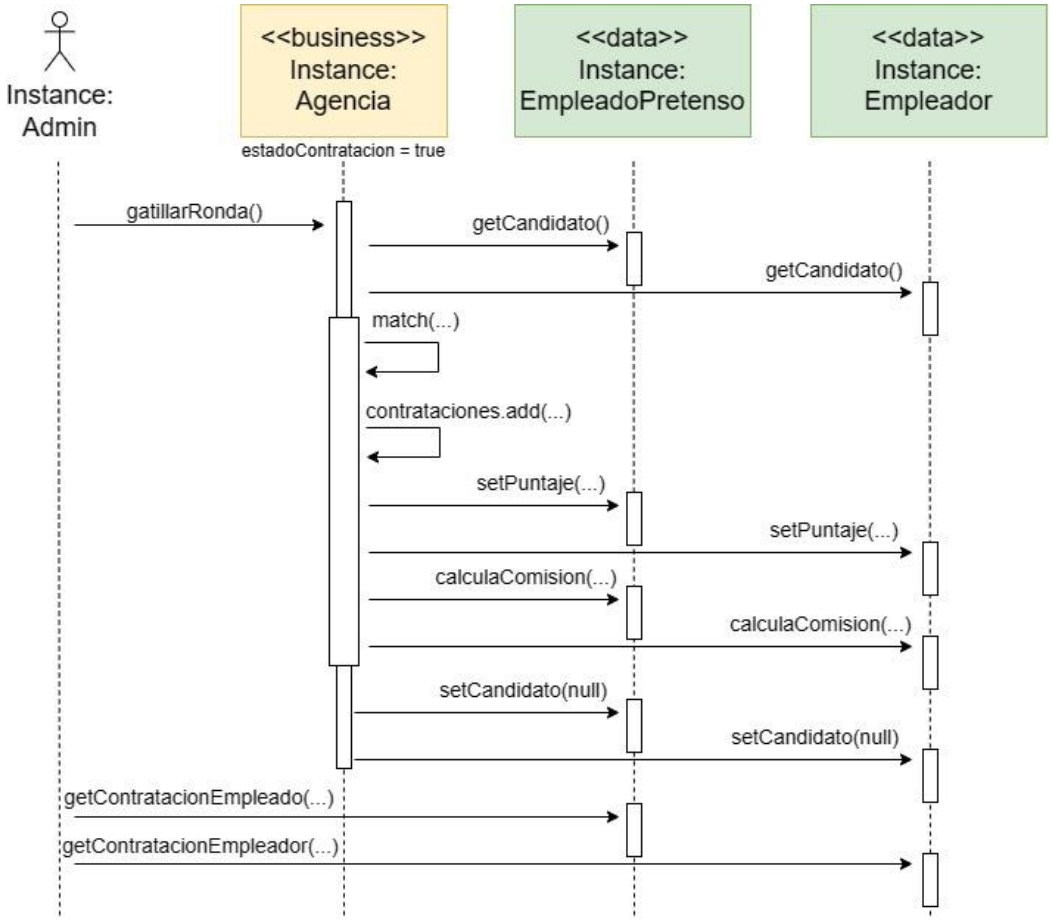
Secuencia 3: Elegir candidato



Secuencia 5: Ronda Encuentros



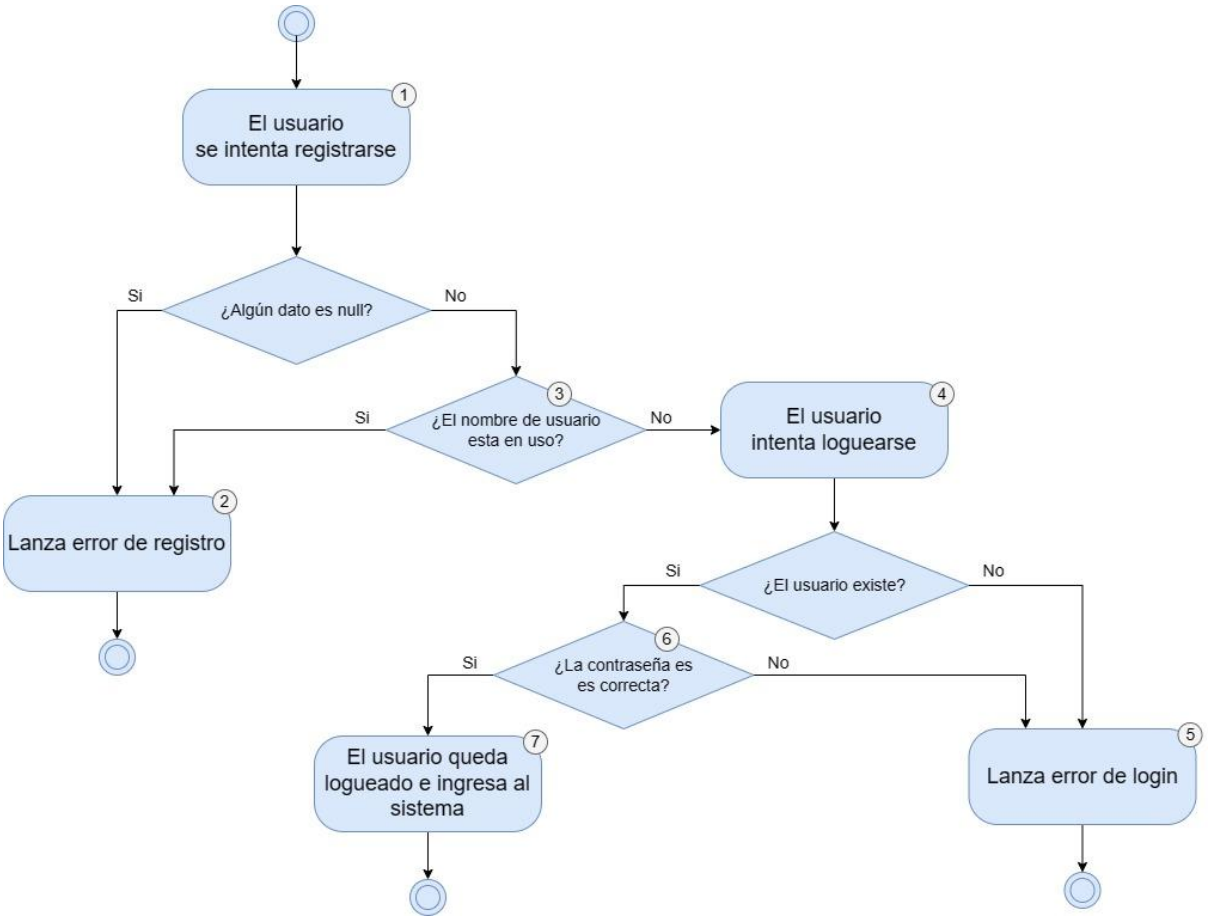
Secuencia 6: Ronda Contratación



Diagramas de Actividad:

Tras el planteo de los casos de uso y sus diagramas de secuencia, se diseñaron y analizaron los diagramas de actividad.
A continuación se presentan los grafos de los diagramas de actividad indicando los caminos a tomar para plantear los casos de prueba.

Grafo 1:



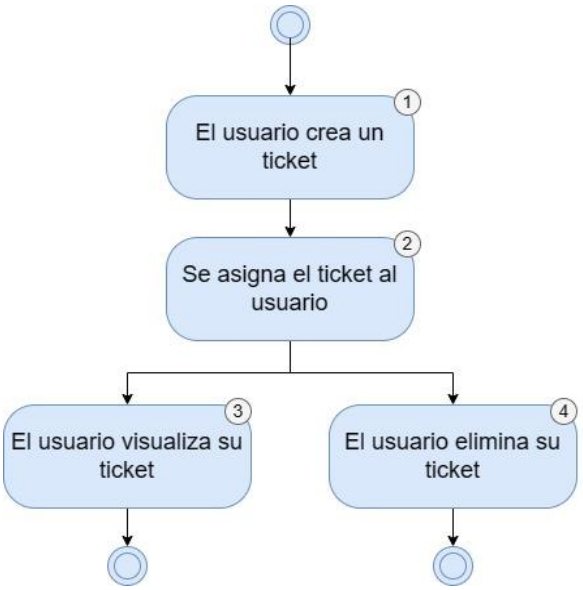
Casos de prueba para Empleado y Empleador:

a) 1, 2
b) 1, 3, 2
c) 1, 3, 4, 5
d) 1, 3, 4, 6, 5
e) 1, 3, 4, 6, 7 (Flujo normal, camino más significativo)

Casos de prueba para Admin:

a) 4, 5
b) 4, 6, 5
c) 4, 6, 7

Grafo 2:

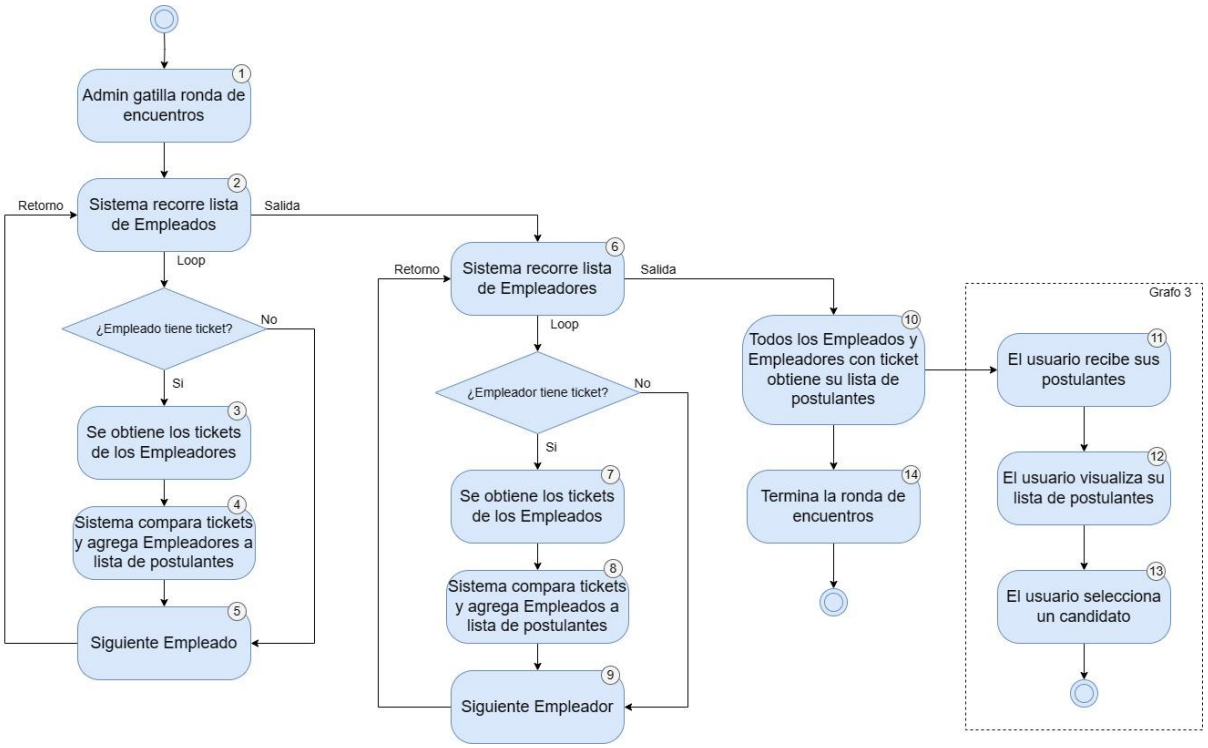


Casos de prueba:

a) 1, 2, 3
b) 1, 2, 4

Grafo 3 y 5:

Tras el análisis de los diagramas de actividad se decidió combinar los casos de uso 3 y 5, pues por sí mismos resultaban simples (principalmente el 3, con un solo camino) y resultaban testeados en su totalidad por los tests de caja negra.
Sin embargo, combinándolos resulta en un test de integración mucho más valioso que, justamente, demuestra la integración entre ambas funcionalidades.



La naturaleza del sistema permite que todo el grafo se pueda contemplar en un escenario, y cada caso de prueba consistirá en loguearse con cada usuario y corroborar que tengan los postulantes esperados.
En el escenario se registran 3 Empleados y 3 Empleadores, donde 1 de cada grupo no tendrá un ticket creado.
Finalmente se espera que los usuarios con tickets puedan ver una lista con los otros dos postulantes y sean capaces de seleccionar a uno como su candidato.

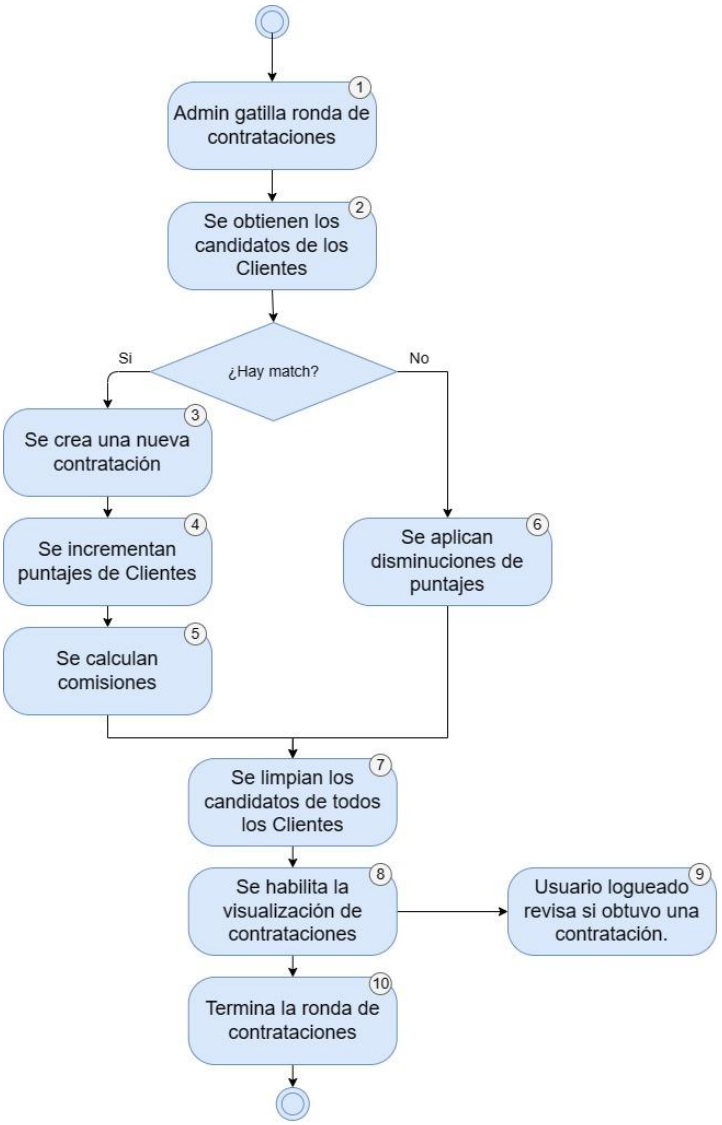
Grafo 6:

De manera similar al caso anterior este grafo se puede testear con un escenario.

El escenario para este caso de prueba es similar al planteado para caja negra:

- Se tiene dos Empleados y 2 Empleadores con tickets.
- El Empleado 1 y 2 setearon como candidato al Empleador 1.
- El Empleador 1 seteo como candidato al Empleado 1.
- El Empleador 2 seteo como candidato al Empleado 2.

La principal diferencia radica en el nodo 9. Los usuarios logueados deben ser capaces de ver sus contrataciones (si es que tienen alguna) junto a cómo se afectaron sus puntajes y comisiones.



Conclusión

Utilizando las técnicas de testing vista en la materia , se pudo detectar varios errores en distintas partes del software, dicho software que en principio no parecía mostrar fallas en su ejecución.
El test más interesante de llevar a cabo fue el de GUI dado que no es muy común en nuestras prácticas testear ventanas
la tarea de testear este software en si no es complicado sino más bien laboriosa, una vez entendido los datos que se deben testear la codificación de los test se vuelve simple y repetitiva.
En lo que respecta al test de caja negra se observó que las clases ticket, empleadoPretenso, empleador y agencias presentan fallos, lo cual implica que realizar el test fue útil para detectar dichos errores, partiendo de implementar buenas baterías de prueba
En lo que respecta al test de caja blanca se observó que los caminos escogidos no presentaron fallos, se logró una cobertura de todos los ciclos y se demuestra que dicha clase no presenta fallas en su implementación
En lo que respecta al test de GUI se observó que el test efectivamente arrojo fallas en el panel admin, cliente , login, cambio y registro, los cuales fueron descubierto con la ayuda de la clase robot, la cual tiene la ventaja de imitar el comportamiento de un usuario, es decir, simula al autocompletado de los campos como si se tratase de una persona proporcionando una cobertura más amplia
En lo que respecta al test de persistencia se utilizó el archivo”persistencia.xml”, toda la aplicación del testeo se hizo teniendo en cuenta no romper con el patrón de diseño singleton del cual hace uso la clase agencia