Universidad de las Ciencias Informáticas

23 de junio de 2016

"Año 58 de la Revolución"

Suite de pruebas automatizadas para el Juez en Línea Caribeño

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores

Wendys de la Caridad Jiménez Estévez
Franly Hernández Chávez

Tutores

Ing. Yudislandry Águila González
Ing. Sandy Guerra Fernández

"Es preciso entender la tesis como una ocasión única para hacer algunos ejercicios que nos servirán mientras vivamos."

Umberto Eco

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Wendys de la Caridad Jiménez Estévez

_____

Tutor: Ing. Yudislandry Águila González

_____

Franly Hernández Chávez

_____

Tutor: Ing. Sandy Guerra Fernández

_____

DEDICATORIA Y AGRADECIMIENTOS

Wendys de la Caridad Jiménez Estévez

A mi novio Jorge Lazaro, a quien agradezco por todo lo que hemos pasado juntos, por la ayuda que me ha brindado en cada momento difícil durante estos cinco años, por obligarme a estudiar para cada examen, por estar siempre que lo he necesitado. Por todo tu amor.

A mi compañero de tesis Franly, por aceptar mi propuesta de hacer la tesis juntos y mantenerse firme a pesar de todas las trabas, por sus chistes y su alegría que nos han hecho reír y sentir bien a más de uno.

A los miembros de mi familia que me han apoyado en todo momento y que han sido mi motor impulsor y mi inspiración para lograr cada cosa que he hecho en la universidad:

- A mi mamá, mi hermano Yoisel, mi abuela Felicia y a Walbe, mi segundo papá. Gracias a todos ustedes por su cariño, su apoyo, por quererme tanto, por convertirme en la mujer que soy y de la cual espero estén muy orgullosos. Muchas gracias a Leobanis por su contribución a mi tesis.

- A mi papá, por ayudarme a lograr mi sueño, por acomodarme y sobre todo, porque siento que desde que convivimos ha aprendido a confiar en mí, a entenderme y a quererme un poco más.

- A mis tíos Ileana y Benito, a mis primos Jorge Félix y Víctor, a Ismaray y a mis hermanos pequeños Luis Miguel y Karla, por abrirme las puertas de su casa y de su corazón, por cubrirme, por ser mis cómplices.

A mis mejores amigos de la universidad: Tania, Edelín, Johancito, Lester, Yaise, Papo, Leduán y a Henrry que no podía faltar, él sabe cómo molestarme y a la vez tranquilizarme con sus palabras.

Al resto de los miembros del grupo, que me acogieron como una más cuando llegué de última en primer año, a los que por un motivo u otro han quedado detrás y a los que llegaron después. Gracias a todos por la buena relación que siempre han tenido conmigo y por su confianza.

A mis compañeros de la ESBU: Yoan, Yazmín, Erne, Rodnier, Eduardo y Daynier. Juntos fuimos capaces de enfrentarnos al mayor de los retos: un grupo de 30 estudiantes de la secundaria "28 de enero". Superamos muchos obstáculos juntos y pasamos inolvidables momentos.

A mis tutores Sandy y Yudislandry, que sirvieron de guía y de apoyo en este semestre durante el desarrollo de la tesis, sus alentadoras palabras que quitaron más de una molestia.

A todos los profesores que me han impartido clases. Les agradezco por todas sus enseñanzas y su paciencia. A otros que de una forma u otra han sido un eslabón clave en mi formación como profesional.

Al resto de mis amistades, tanto en la universidad como fuera de ella, que estuvieron siempre pendiente del avance de mi tesis, de mi estancia en la UCI, que cada vez que me veían me decían puros halagos.

A todos y cada uno de ustedes, gracias por todo su apoyo.

DEDICATORIA Y AGRADECIMIENTOS

Franly Hernández Chávez

En este momento tan especial en el que logro alcanzar uno de mis sueños me siento agradecido a muchas personas, que de una forma u otra estuvieron presentes a lo largo del camino. Personas que con perseverancia, astucia, dedicación y algunas que sin quererlo lograron que creciera como persona y contribuyeron a convertirme en todo un profesional.

Quiero agradecer y dedicar a mi familia, por ser mi razón de ser, mi mayor tesoro. Gracias por estar siempre que los necesite, por apoyarme en todas mis decisiones, por ese ánimo que me dieron en los momentos más difíciles. Este título es por ustedes y para ustedes. Gracias por todo el sacrificio que tuvieron que hacer para apoyarme en mi sueño, por todo lo que me han querido y enseñado. Abuelo Chávez, esto es para ti también, que a pesar de no estar conmigo físicamente, está presente en cada paso que doy, seguro que estarías muy orgulloso. La vida no me alcanzaría para agradecer tan valioso regalo, una familia como ustedes.

A mi novia Greicel, por todo el amor, cariño, comprensión y apoyo brindado. Gracias por estar ahí cada vez que lo necesite, por darme fuerzas y ánimo para seguir adelante. Por ser mí amiga, compañera y confidente. Por llegar a mi vida en el momento preciso.

A mi compañera de tesis Wendys, gracias por soportarme y ayudar a convertir en realidad mi sueño, sin ti no lo hubiera conseguido.

A mis amigos, los viejos y los que conocí en la universidad, por estar siempre conmigo, por los consejos brindados y los momentos compartidos juntos. Sin ustedes el transcurso por la Universidad no hubiese sido el mismo, los voy a extrañar y a recordar como mi segunda familia.

A mis tutores por su preocupación y apoyo en cada paso en la realización de esta tesis.

Al tribunal y oponente pues con sus señalamientos hicieron posible alcanzar la calidad de la tesis.

A todos los profesores que durante cinco años contribuyeron a mi formación profesional e integrar. Gracias por todos los conocimientos brindados y por todos los valores que me inculcaron.

Resumen

Los jueces en línea son aplicaciones web que proveen un espacio donde personas de todo el mundo intercambian sus habilidades en programación. Un ejemplo claro de este tipo de aplicaciones es el Juez en Línea Caribeño (COJ, por sus siglas en inglés), desarrollado por un conjunto de programadores de la Universidad de las Ciencias Informáticas y otros pertenecientes a la comunidad caribeña del ACM International Collegiate Programming Contest (ACM-ICPC). El presente trabajo tiene como objetivo la implementación de una suite de pruebas automatizadas para el COJ que contribuya a la ejecución de actividades para corregir sus defectos, debido a los resultados arrojados por el estudio de los principales problemas que impiden efectuar el mantenimiento de la aplicación de forma más factible, identificándose como deficiencias fundamentales la dificultad en la detección de errores en el código y la rapidez de su corrección. Debido a lo anterior, es necesario contribuir al mantenimiento del sistema, para lo cual se analizaron e implementaron un conjunto de pruebas utilizando diversas herramientas. La validación de la propuesta estuvo fundamentada sobre la técnica de Iadov, en la cual se realizó una valoración del nivel de satisfacción de la solución mediante la realización de una encuesta a un conjunto de personas relacionadas con el sistema.

Palabras clave: jueces en línea, Juez en Línea Caribeño, suite de pruebas automatizadas, técnica de Iadov.

Índice general

Contenido

Índice de tablas

Índice de ilustraciones

Índice de ecuaciones

Introducción

La programación indudablemente tiene gran importancia en las carreras de ingeniería o de perfil computacional. El objetivo de enseñar esta materia es lograr que los estudiantes sean capaces de resolver problemas de la vida cotidiana surgidos durante su período de formación y posteriormente en su labor como profesional, así como desarrollar en ellos la capacidad de abstracción como base para el análisis de problemas sencillos y el diseño posterior de algoritmos que los resuelvan.

Una de las estrategias tomadas con el objetivo de incidir en el aprendizaje de la materia señalada ha sido el desarrollo de los concursos de programación. Este tipo de eventos surge en los años 70 por iniciativa del profesor Bill Poucher de la Universidad de Baylor, en Estados Unidos de América, quien se desempeña actualmente como director ejecutivo del ACM International Collegiate Programming Contest (ACM-ICPC).

El concurso ACM es efectuado anualmente con representación de diferentes universidades del mundo. Este constituye en la actualidad uno de los más importantes y prestigiosos eventos de programación de computadoras, donde se fomenta el trabajo en equipo, la resolución de tareas, la creatividad y la innovación en la creación de programas de software, poniendo a prueba la capacidad de los estudiantes para la búsqueda de soluciones a disímiles problemas.

Para brindar soporte a la preparación de esos concursos surgen los jueces en línea, definidos como aplicaciones web creadas principalmente para entornos académicos, que proveen problemas de variadas complejidades, y a su vez, evalúan de forma automática las soluciones que los usuarios encuentran a los problemas. En los últimos años se ha potenciado su uso por parte de estudiantes y entrenadores con el propósito de lograr buenos resultados en este tipo de eventos.

Debido al auge del ACM-ICPC, se ha incentivado durante los últimos años (2006 – 2012) en la Universidad de las Ciencias Informáticas (UCI) la programación de competencia como disciplina formativa y educacional, mediante la creación de un movimiento de programación competitiva. Este ha tenido un crecimiento constante durante todos los años de existencia de la misma a través de diferentes vertientes y grupos, que concluyó finalmente con la aceptación de Cuba en el movimiento mundial ACM-ICPC.

Son varios los años de experiencia acumulados, durante los cuales se ha mejorado la organización y la cantidad de personas involucradas, culminando con la creación del Caribbean Online Judge (COJ), primer juez en línea cubano, disponible en Internet desde el 5 de junio de 2010 a través de la dirección electrónica http://coj.uci.cu.

El sistema provee un espacio donde personas de todo el mundo intercambian experiencias y conocimientos, así como prueban, mejoran y comparten sus problemas y soluciones, además sus habilidades en programación, por lo que sus usos más comunes son:

- ✓ Como herramienta educativa para instrucción formal.
- ✓ Como herramienta de autoaprendizaje y autosuperación.
- ✓ Como sistema de manejo de competencias.
- ✓ Como entrenamiento.

La corrección de los defectos en el COJ es de gran interés debido fundamentalmente a su importancia en el país, pues permite que todos los aficionados, profesionales y estudiantes cubanos tengan una solución (producto) nacional bajo el dominio .cu, lo que aumenta la accesibilidad para la comunidad de programadores y concursantes de Cuba y contribuye al desarrollo del sector informático.

Debido a lo anterior, se lleva a cabo la evaluación y verificación del correcto funcionamiento de dicho sistema, observando directamente su comportamiento y depurando las deficiencias cada vez que aparecen. Este proceso es denominado pruebas de software y permite verificar la calidad de un software, así como determinar el grado de cumplimiento de las expectativas esperadas, por lo que es indispensable y debe ser desarrollado con muchísima frecuencia, comprobando que con el transcurso del tiempo las nuevas funcionalidades que han sido incorporadas den total respuesta a las exigencias de los clientes. Estas pruebas que se realizan de forma manual, son parte de la etapa de mantenimiento del software, que es un conjunto de acciones para corregir defectos, mejorar el rendimiento o adaptar el sistema a un cambio en el entorno.

Actualmente es casi imposible conseguir un software totalmente libre de defectos y el COJ no constituye la excepción, ya que es probable que se originen algunas fallas durante su ejecución, por tanto sería preciso que el encargado de llevar a cabo el mantenimiento tenga conocimiento acerca de dicho sitio web en su totalidad, ya que resultaría difícil la localización de errores y su modificación, hasta pudieran llegar a ser necesarias más de una persona para realizarlo. Todo lo anterior puede implicar una pérdida de tiempo considerable, incluso porque no se puede asegurar lo que demoraría ejecutar la fase.

Además, las pruebas manuales que se realizan en el COJ resultan largas y complejas, ya que no existe un mecanismo que permita comprobar de forma automática el correcto funcionamiento de sus módulos, lo que puede representar una limitación cuando se

desea conocer el número o la importancia de los errores que se detectan en su programación.

Un claro ejemplo que evidencia las consecuencias de las deficiencias antes mencionadas sería la existencia de errores en uno de los paquetes del COJ, para lo que se hace inevitable la revisión de cada módulo tratando de detectarlos pero no hay garantía de lograr el objetivo.

Por tal motivo se centrarán los esfuerzos en darle solución al siguiente **problema**: ¿Cómo contribuir al proceso de pruebas que se realiza como parte de la etapa de mantenimiento del Juez en Línea Caribeño?

A partir del problema a resolver se puede inferir que el **objeto de estudio** son las pruebas de software en aplicaciones web, teniendo como **campo de acción** las pruebas automatizadas de software para el Juez en Línea Caribeño.

Para dar respuesta al problema anterior se definió como **objetivo general** desarrollar una suite de pruebas automatizadas para el Juez en Línea Caribeño que contribuya al proceso de pruebas que se realiza como parte de la etapa de mantenimiento.

En correspondencia con ello se plantean los siguientes **objetivos específicos**:

- ✓ Establecer los referentes teóricos y metodológicos relacionados con las pruebas de software, haciendo énfasis en las pruebas automatizadas.
- ✓ Seleccionar adecuadamente las herramientas que facilitarán la creación de pruebas automatizadas para el COJ.
- ✓ Desarrollar la suite de pruebas automatizadas para el COJ.
- ✓ Valorar la efectividad del uso de las pruebas automatizadas de software para el mantenimiento del COJ.

Las **tareas** definidas para dar cumplimiento a los objetivos trazados son**:**

- ✓ Revisión de la documentación relacionada con las pruebas automatizadas y el COJ.
- ✓ Selección del tipo de prueba automatizada a desarrollar en la suite.
- ✓ Diseño y descripción de las pruebas que se implementarán.
- ✓ Selección de las herramientas y el lenguaje de programación a utilizar, teniendo en cuenta los esquemas del proceso integral de desarrollo del COJ.
- ✓ Desarrollo de la suite de pruebas automatizadas a partir de los casos de prueba del sitio web.

- ✓ Realización del proceso de validación para verificar la completitud y la corrección de las pruebas.

Los **métodos científicos** utilizados en la investigación son:

Teóricos:

- ✓ Analítico-sintético: permite comprender los aspectos relacionados con las pruebas automatizadas, así como realizar el análisis e interpretación de los resultados obtenidos.
- ✓ Histórico-lógico: utilizado para analizar la evolución histórica y desarrollo de las pruebas automatizadas en el COJ.
- ✓ Modelación: utilizado para la caracterización de las funcionalidades que tendrá el componente.

Empíricos:

- ✓ Observación: utilizado para observar las deficiencias de mantenimiento que presenta el COJ.
- ✓ Encuesta: utilizado para obtener información a partir de las respuestas de varias personas a cuestionarios pre elaborados con el objetivo de colaborar con la investigación.

Métodos cualitativos y cuantitativos:

- ✓ Los modelos de análisis cuantitativos para el procesamiento de los datos y análisis de las encuestas.

Métodos matemáticos:

- ✓ La estadística descriptiva para el análisis de los resultados.

Se empleó además la Técnica de Iadov para conocer el nivel de satisfacción de los clientes con el sistema propuesto.

La aplicación sistemática de estos métodos, permitió alcanzar los resultados previstos en cada una de las etapas de la investigación.

El contenido de la investigación está estructurado en tres capítulos:

Capítulo 1: Fundamentación Teórica: en el capítulo 1 se analizan diferentes conceptos como punto de partida para la fundamentación. Se realiza un estudio de las pruebas de software, haciendo énfasis en las pruebas automatizadas. Además se analizan las principales herramientas que serán usadas en el desarrollo de la suite.

Capítulo 2: Desarrollo de la solución propuesta de solución: se exponen las herramientas seleccionadas y las razones de su elección para conformar la suite de pruebas de acuerdo a las características del software en desarrollo.

Capítulo 3: Implementación y validación: se describirá cómo se ha aplicado el proceso teniendo en cuenta las herramientas propuestas en el capítulo 2 y mediante la técnica de Iadov se realizará la validación del sistema, a partir de los resultados obtenidos en una encuesta de satisfacción.

Capítulo 1: Fundamentación teórica

El presente capítulo tiene como objetivo fundamental, profundizar en diferentes temas que se utilizarán como soporte teórico para el entendimiento de la solución que se propone. Se presentan los conceptos fundamentales relacionados con el tema en cuestión y posteriormente se analizan algunas de las herramientas y tecnologías más destacadas en el ámbito del desarrollo de sistemas informáticos, seleccionando aquellas que serán utilizadas en la etapa de desarrollo.

1.1 Pruebas de software

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error, no puede asegurar la ausencia de defectos, sino demostrar la presencia de los mismos en el software, por lo que constituye el único instrumento adecuado para determinar el status de la calidad de un producto. (1) En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema en su totalidad, con el objetivo de medir el grado en que este cumple con los requerimientos. En ellas se usan casos de prueba, especificados de forma estructurada mediante diferentes técnicas.

Ilustración 1. Proceso general de pruebas. (2)

Entre sus objetivos están:

- ✓ Verificar que todos los requisitos se han implementado correctamente.
- ✓ Detectar defectos en el software.
- ✓ Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- ✓ Diseñar casos de prueba que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo en el menor tiempo y esfuerzo posible.

Las pruebas se rigen por una serie de principios, una buena comprensión de estos facilitará el posterior uso de los métodos en un efectivo diseño de casos de prueba. A continuación se citan:

- ✓ La prueba puede ser usada para mostrar la presencia de errores, pero nunca su ausencia.
- ✓ La principal dificultad del proceso de prueba es decidir cuándo parar.
- ✓ Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.
- ✓ Una parte necesaria de un caso de prueba es la definición del resultado esperado.
- ✓ Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
- ✓ El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.

Estas leyes que definen básicamente la aplicación de las pruebas de software ayudan a refinar el producto de software a través de las etapas involucradas:

1. Seleccionar qué es lo que debe medir la prueba, es decir, cuál es su objetivo, para qué exactamente se hace la prueba.
2. Decidir cómo se va a realizar la prueba, es decir, qué clase de prueba se va a utilizar para medir la calidad y qué clase de elementos de prueba se deben usar.
3. Desarrollar los casos de prueba, que no son más que un conjunto de datos o situaciones de prueba que se utilizarán para ejecutar la unidad que se prueba o para revelar algo sobre el atributo de calidad que se está midiendo.
4. Determinar cuáles deberían ser los resultados esperados de los casos de prueba y crear el documento que los contenga.
5. Ejecutar los casos de prueba.

Según su ejecución las pruebas se clasifican como:

Manuales: son las que se hacen normalmente al programar o las que ejecuta una persona con la documentación generada durante la codificación (Por ejemplo: comprobar cómo se visualiza el contenido de una página web en dos navegadores diferentes). (3)

Automatizadas: consiste en el uso de software especial (casi siempre separado del software que se prueba) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados.

1.1.1 Comparación entre pruebas manuales y pruebas automatizadas

Las pruebas manuales de software se realizan por alguna persona (ya sea cliente, desarrollador, probador, entre otros) que interactúa con la interfaz de la aplicación, verificando si los resultados obtenidos son los deseados mediante juegos de datos. Son realizadas constantemente, debido a errores detectados en el software, cambios en los requisitos, entre otros factores.

Sin embargo, una herramienta de pruebas automatizadas de software es capaz de reproducir grabados y acciones predefinidas, comparar los resultados con el comportamiento esperado y reportar el éxito o el fracaso de estas pruebas. Una vez que las pruebas automatizadas se crean se pueden repetir y ampliar para realizar tareas que serían imposibles con las pruebas manuales. Debido a esto, las empresas más desarrolladas y exitosas han encontrado que las pruebas de software automatizadas son un componente esencial para desarrollar software de elevada calidad.

Las pruebas automatizadas son superiores a las manuales en varios aspectos:

Las pruebas automáticas resultan muy cómodas para ejecutarlas con bastante frecuencia. Lo más usual es que cada vez que se agregue código al software, se ejecuten dichas pruebas. Si el proceso genera errores, se podrá identificar fácilmente que el nuevo código que se ha introducido tiene algún fallo. Si la prueba es larga y manual, debido al costo de esta, la frecuencia disminuye. Mientras más pasa el tiempo sin ejecutar las pruebas, la cantidad de código del proyecto aumenta, por lo que la búsqueda de un fallo se hará mucho más engorrosa y se consumirá más tiempo.

Las pruebas automáticas dan seguridad al equipo de desarrollo para perfeccionar o arreglar el código que ya está funcionando. En ocasiones el código está funcionando pero no está del todo bien. Otras veces, es necesario cambiar un poco ese código para poder reutilizarlo en una nueva funcionalidad que se quiera agregar al proyecto. Si no existe una forma fácil de ver que ese código sigue funcionando después de cambiarlo, hacerle cambios sería un gran riesgo, con consecuencias desastrosas para el desarrollo del software, usualmente se decide no hacerlo.

Una prueba automática que falla brinda más información que una prueba manual que detecta un fallo. Si una prueba automática falla, dice exactamente qué método del código no hace lo que se esperaba de él. Si una prueba manual encuentra un fallo, sólo se conoce que algo en cualquier parte del código no está bien.

1.2 Pruebas automatizadas de software

En un entorno de integración continua, las pruebas se ejecutan de manera regular, con el objetivo de probar que la evolución del software desarrollado funciona perfectamente, tanto para las funcionalidades nuevas como para aquellas que ya existían. Una de las claves es que las pruebas sean automatizadas, porque si se ejecutaran manualmente, el coste de hacerlo continuamente sería inabordable.

La automatización de pruebas consiste en el uso de software especial para controlar su ejecución y realizar una comparación entre los resultados obtenidos y los esperados. Permite incluir pruebas repetitivas y necesarias dentro de un proceso ya existente, o bien adicionar otras cuya ejecución manual resultaría difícil, de este modo se crea un primer criterio por el cual seleccionar una prueba y otras no. Es decir, el proceso de automatización se realiza para agilizar las actividades de testeo de una aplicación, haciéndolo de carácter selectivo en cuanto a las pruebas a automatizar, siendo estas las que sean más frecuentemente aplicadas o difíciles de testear de manera manual. (4)

1.2.1 Características de las pruebas automatizadas

En resumen las pruebas automatizadas se caracterizan por ser (5):

Confiables

- ✓ Las pruebas automatizadas realizan las mismas operaciones que las manuales con una precisión mayor cada vez que se ejecutan, eliminando el error humano.

Repetibles

- ✓ Brindan la posibilidad de probar cómo reacciona el software en ejecución repitiendo las mismas operaciones de pruebas.

Programables

- ✓ Permiten programar pruebas sofisticadas que ponen en evidencia la información oculta de la aplicación.

Integrales

- ✓ Permiten construir un conjunto de pruebas que cubre todas las características de la aplicación a probar.

Reutilizables

- ✓ Las pruebas automatizadas una vez construidas pueden volverse a utilizar sobre las diferentes versiones de una aplicación, incluso si cambia la interfaz de usuario.

Rápidas

- ✓ Las pruebas se ejecutan a través de herramientas automatizadas, las cuales son significativamente más rápidas que los usuarios humanos.

Reducción de Costos

- ✓ El número de recursos se reducen considerablemente, tanto materiales como humanos.

1.2.2 Ventajas de la automatización de pruebas

Algunas características de las pruebas automatizadas son: (6)

Ahorran tiempo y dinero

Las pruebas de software tienen que ser repetidas con frecuencia durante el ciclo de desarrollo para asegurar la calidad. Cada vez que se modifica el código fuente las pruebas de software se deben repetir. Cada versión del software puede ser probada en diferentes sistemas operativos y configuraciones de hardware. Repetir las pruebas manuales es costoso y consume tiempo. Sin embargo, una vez creadas, las pruebas automatizadas se pueden ejecutar una y otra vez, sin costo adicional y son mucho más rápidas que las pruebas manuales. Las pruebas automatizadas de software pueden reducir el tiempo para ejecutar las pruebas repetitivas de días a horas. Un ahorro de tiempo que se traduce directamente en ahorro de costos.

Precisión y exactitud

Debido a la monotonía de las pruebas manuales es muy frecuente que los probadores cometan algunos errores. Las pruebas automatizadas pueden realizar los mismos pasos con más precisión y seguridad cada vez que se ejecutan, registrando los resultados detalladamente.

Calidad del software

Las pruebas automatizadas de software pueden aumentar el alcance de las pruebas para ayudar a mejorar la calidad del software. Las pruebas extensas que se han evitado en el uso de las pruebas manuales se pueden ejecutar sin la necesidad de atención del probador. Incluso se pueden ejecutar en varios equipos con configuraciones diferentes.

Las pruebas automatizadas de software pueden buscar dentro de una aplicación y ver contenido de la memoria, las tablas de datos, el contenido de los archivos, y los estados internos del programa para determinar si el producto se está comportando como se esperaba. Las pruebas automatizadas de software pueden ejecutar miles de diferentes casos de pruebas complejas en cada ejecución que es imposible con las pruebas manuales. Los probadores liberados de las pruebas manuales repetitivas tienen más tiempo para crear nuevas pruebas de software automatizadas y hacer frente a las nuevas y complejas características.

Potencialidad

Los más grandes departamentos de software no pueden realizar una prueba de aplicaciones web controladas con miles de usuarios. Las pruebas automatizadas pueden simular decenas, cientos o miles de usuarios virtuales que interactúan con la red o la web, y con la aplicación.

Facilitan el trabajo a los desarrolladores y probadores

Las pruebas automatizadas pueden ser utilizadas por los desarrolladores para detectar los problemas rápidamente. Pueden ejecutarse automáticamente cada vez que los cambios de código fuente se comprueban y notificar al equipo o al desarrollador si fallan.

Aumentan la moral de equipo de trabajo

Según estudios realizados por empresas de desarrollo de software las pruebas de software automatizadas pueden mejorar la moral del equipo. Automatizar tareas repetitivas le brinda al equipo la posibilidad de invertir tiempo en proyectos más desafiantes y gratificantes. Los miembros del equipo tienden a mejorar sus habilidades y confianza y, a su vez, pasar esos beneficios a su organización.

1.2.3 Tipos de pruebas automatizadas

Existen diferentes tipos de pruebas: (7)

Unitarias: se encargan de probar una clase en concreto, testeando cada uno de sus métodos y viendo si dados unos parámetros de entrada, la salida es la esperada.

Funcionales: prueban una funcionalidad completa, donde pueden estar implicadas una o varias clases y la propia interfaz de usuario.

De regresión: son aquellas cuyo objetivo es comprobar por qué ha dejado de funcionar algo que ya funcionaba.

De aceptación: en esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que su uso se justifique.

De integración: conjunto de pruebas unitarias, funcionales, de regresión y/o de aceptación que se realizan para probar el software. Incluye también comprobar que lo programado por los diferentes desarrolladores no "choca" entre sí y que funcionará en un entorno real.
A continuación se hace referencia de forma más específica a cada una de ellas.

Pruebas unitarias

Las pruebas unitarias validan que un objeto aislado del resto del sistema funciona como según lo esperado, es decir, cada módulo será probado por separado y lo hará, generalmente, la persona que lo creó. Se ejecutan con mayor rapidez, pues las herramientas a utilizar solo tienen que instanciar el objeto y aplicarlas. (8)

Un sistema está conformado por distintos objetos que colaboran y para proveer un entorno donde los resultados sólo dependan del componente que se está probando, es necesario apoyar este tipo de pruebas con técnicas como dobles (mocks). Todos los colaboradores externos deben ser "simulados" a través de un doble.

Algunos de los requisitos que deben cumplir las pruebas unitarias para que sean eficientes son:

- ✓ Se tienen que poder ejecutar sin necesidad de intervención manual, lo cual posibilita que se pueda automatizar su ejecución.
- ✓ Deben cubrir casi la totalidad del código de la aplicación.
- ✓ La ejecución de una prueba no puede afectar la ejecución de otra.
- ✓ Las diferentes relaciones que puedan existir entre módulos deben ser simuladas para evitar dependencias entre ellos.
- ✓ Conocer claramente cuál es el objetivo de la prueba.

Las pruebas unitarias son perfectas para probar componentes que representen la lógica de negocio. Algunos de sus beneficios son los siguientes: (9)

- ✓ *Fomentan el cambio*, las pruebas unitarias facilitan la reestructuración del código, puesto que permiten hacer pruebas sobre los cambios y verificar que las modificaciones no han introducido errores.

- ✓ *Simplifican la integración*, permiten llegar a la fase de integración asegurando que las partes individuales funcionan correctamente. De esta manera se facilitan las pruebas de integración.
- ✓ *Documentan el código*, las propias pruebas pueden considerarse documentación, ya que las mismas son una implementación de referencia de cómo utilizar el código.
- ✓ *Separación de la interfaz y la implementación*, la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro (pruebas mock).
- ✓ *Menos errores y más fáciles de localizar*, las pruebas unitarias reducen la cantidad de errores y el tiempo en localizarlos.
- ✓ *Pueden mejorar el diseño*, la utilización de prácticas de diseño y desarrollo dirigida por las pruebas (Test Driven Development o TDD) permite definir el comportamiento esperado en un paso previo a la codificación.
- ✓ *Puede ser la forma más simple de verificar el funcionamiento*, en situaciones como el desarrollo de un componente que brinda servicios del cual no se cuenta aún con un cliente para consumirlos.

Pruebas Mock: pruebas unitarias que utilizan objetos simulados ("mock") que sustituyen a los objetos reales utilizados por la clase o fragmento de código a probar. Su principal ventaja es que reducen drásticamente el número de líneas de código de los tests de validación y de interacción. También se usan cuando hay que acceder a datos precedentes de un colaborador. Por lo tanto los mocks y otros dobles son imprescindibles para un desarrollo dirigido por tests completos, pero igualmente importante es saber cuándo evitarlos.

Pruebas de integración

Aun cuando los módulos de un programa funcionen bien por separado es necesario probarlos conjuntamente: un módulo puede tener un efecto adverso o inadvertido sobre otro módulo. Las subfunciones, cuando se combinan, pueden no producir la función principal deseada; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables al combinar los módulos; los datos pueden perderse o malinterpretarse entre interfaces, etc. (10)

Por tal motivo, es necesario probar el software ensamblando todos los módulos probados previamente, siendo este el objetivo de las pruebas de integración, las cuales

se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias.

En la fase de pruebas de integración los módulos individuales de software son combinados y probados como un grupo. Son las pruebas posteriores a unitarias y preceden a las de regresión.

Para la realización de este tipo de pruebas se utilizan algunas técnicas: (11)

- ✓ *Pruebas de Caja Blanca o Estructurales:* A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento. Su principal objetivo es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa.

- ✓ *Pruebas de Caja Negra:* También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una "Caja Negra" cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable se selecciona un conjunto de ellas sobre las que se realizan las pruebas.

Pruebas de regresión

Cuando se resuelve un error se está modificando el código; y esto tiene un riesgo evidente: que deje de funcionar algo que previamente lo hacía. Por ese motivo existen las pruebas de regresión.

Se denominan pruebas de regresión a cualquier tipo de prueba de software que intenta descubrir errores (bugs), carencias de funcionalidad o divergencias funcionales con respecto al comportamiento esperado del software. Pueden ser causados por la realización de un cambio en el programa, ya sea debido a prácticas no adecuadas del control de versiones, a la falta de consideración acerca del contexto de producción final, a la extensibilidad del error que fue corregido (fragilidad de la corrección) o simplemente una consecuencia del rediseño de la aplicación. Generalmente se ejecutan en cada uno de los pasos del ciclo de vida del desarrollo del software. (8)

No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes. Pueden incluir: la repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada; y la revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.

Tipos de regresión:

- ✓ Clasificación de ámbito:
  - Local (los cambios introducen nuevos errores).
  - Desenmascarada (los cambios revelan errores previos).
  - Remota (los cambios vinculan algunas partes del programa (módulo) e introducen errores en ella).
- ✓ Clasificación temporal:
  - Nueva característica (los cambios realizados con respecto a nuevas funcionalidades en la versión, introducen errores en otras novedades en la misma versión del software).
  - Característica preexistente (los cambios realizados con respecto a nuevas funcionalidades, introducen errores en las ya existentes en previas versiones).

En las pruebas de regresión se debe:

- ✓ Probar íntegramente los módulos que se han cambiado.
- ✓ Decidir las pruebas a efectuar para los módulos que no han cambiado y que han sido afectados por los cambios producidos.

Este tipo de pruebas ha de realizarse tanto durante el desarrollo cuando se produzcan cambios en el software, como durante el mantenimiento.

Pruebas funcionales

Las pruebas funcionales están basadas en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático, en otras palabras, son específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y sobre todo, lo que se ha especificado, es decir, su principal objetivo es objetivo validar si el software probado cumple o no con sus especificaciones. (12)

Dentro de los principales beneficios que tienen este tipo de pruebas está la mitigación del riesgo de aparición de fallos en producción, el cumplimiento de los objetivos de los proyectos en términos de calidad y resultados esperados principalmente, pero también de plazos y costos. Además, la identificación temprana de riesgos y desviaciones asociadas a la calidad, permiten evitar problemas con proveedores, mayores costos para el cliente y finalmente generar confianza en el producto o sistema bajo test. (13)

Se dividen en las siguientes fases:

1. *Análisis de requisitos (Planificación)*

   En la planificación se inicia la elaboración del modelo jerárquico de requisitos de prueba, partiendo de los procesos funcionales soportados por el producto o activo de software a evaluar. A partir de las funcionalidades se elaborará el plan de pruebas. Hay que obtener toda la información posible de las aplicaciones sobre las cuales se realizarán las pruebas. Esta información se deberá conseguir de toda la documentación disponible sobre su funcionamiento y hablando con el personal responsable de la misma.

2. *Diseño del plan de pruebas (Preparación)*

   La preparación consiste fundamentalmente en identificar, acordar y especificar los atributos de calidad que se van a probar. El objetivo es diseñar las pruebas para que tengan la mayor probabilidad de encontrar defectos con la mínima cantidad de esfuerzo y tiempo. Serán pruebas que se llevarán a cabo a través de la interfaz gráfica del software. Se crearán casos de prueba divididos en pasos (steps) para cada acción a realizar con un resultado esperado asociado, que podrá ser verificado. También se especifican los datos de entrada necesarios para que los casos de pruebas definidos puedan ser ejecutados.

3. *Ejecución*

   En esta fase se ejecutan los casos de prueba anteriormente diseñados de forma manual, siguiendo al detalle el guion establecido dejando cierta libertad al tester para detectar situaciones anómalas no contempladas. Las pruebas serán ejecutadas como mínimo una vez antes del paso a producción, independientemente de las ejecuciones anteriores. Los casos de prueba fallados se reportarán a los desarrolladores para su corrección hasta que su resultado sea correcto.

4. *Gestión de Incidencias (Defectos)*

   La gestión de incidencias es una parte implícita de la fase de ejecución, pero debido a su importancia en las pruebas funcionales, se diferencia como una etapa

independiente. Si al realizarla el resultado obtenido no es el esperado, habrá que abrir o reportar una incidencia para que el equipo de desarrollo tenga constancia del error. Estas han de ser claras y detalladas, tienen que describir el error para que pueda ser comprendido, reproducido, localizado y solucionado perfectamente. Se deberá mantener una continua comunicación con los desarrolladores para conocer el estado de los defectos y poder realizar las repruebas necesarias para su cierre.

Las pruebas funcionales pueden ser, según su ejecución:

Manuales: son las que ejecuta un tester como si fuese un usuario pero siguiendo una serie de pasos establecidos, diseñados en el análisis de los requisitos para garantizar que hace lo que debe (casos positivos), que no falla (casos negativos) y que es lo que se ha solicitado.

Automáticas: se automatizan para "ahorrar tiempo de pruebas". A partir de los casos de prueba de las pruebas manuales, se automatizan los casos de prueba que se repitan en las ejecuciones.

Pruebas de aceptación

La idea de automatizar las pruebas de usuario es una práctica que se origina con XP (Extreme Programming). En lugar de que el usuario de negocio pase los requerimientos al equipo de desarrollo sin mayor oportunidad de retro alimentarlo, el desarrollador y el usuario colaboran para escribir las pruebas automatizadas que expresen lo que el usuario de negocio requiere. Esto es denominado pruebas de aceptación, pues expresan lo que el software necesita realizar de forma que el usuario de negocio acepte dicha funcionalidad. El test inicialmente falla al ser escrito, pues no hay código relacionado, pero captura lo que el usuario de negocio necesita y proporciona información clara de lo que significa *hecho* para este. (14)

Estas pruebas son diferentes a las pruebas unitarias, ya que estas últimas aseguran la corrección de lo que se construye, mientras que las pruebas de aceptación aseguran que lo que se está construyendo correctamente lo que se necesita. (14)

Sin la participación del usuario de negocio en las pruebas de aceptación, es difícil para los programadores saber qué pruebas necesitan escribir. Las pruebas de aceptación ayudan al equipo a enfocarse, asegurando que el trabajo realizado en cada iteración presenta el mayor valor posible a ser entregado. Igualmente se pueden cometer errores, pero de esta forma se minimizan. (14)

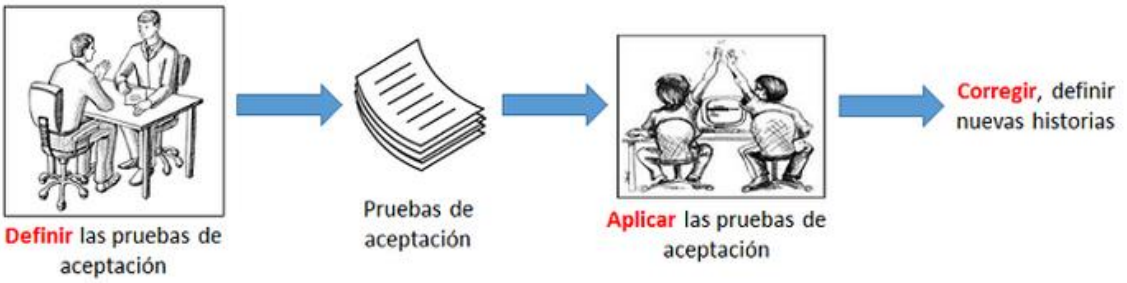

Ilustración 2. Automatización de pruebas de aceptación.

Adicional a su propósito fundamental, las pruebas de aceptación también pueden ser utilizadas para: (15)

- ✓ Obligar a definir requisitos que sean verificables.
- ✓ Valorar adecuadamente el esfuerzo asociado a la incorporación de un requisito.
- ✓ Negociar con el cliente el alcance del sistema.
- ✓ Planificar el desarrollo iterativo e incremental del sistema.
- ✓ Guiar a los desarrolladores.
- ✓ Identificar oportunidades de reutilización.

1.3 Mantenimiento de software

El estándar IEEE 1219 [IEEE, 1993] define el Mantenimiento del Software como "la modificación de un producto software después de haber sido entregado [a los usuarios o clientes] con el fin de corregir defectos, mejorar el rendimiento u otros atributos, o adaptarlo a un cambio en el entorno". Dicha etapa se considera una de las fases en el ciclo de vida de desarrollo de sistemas (SDLC, sigla en inglés de *System Development Life Cycle*) y se realiza después del despliegue (implementación) del software. (16) (17)

En el estándar ISO 12207, de Procesos del Ciclo de Vida del Software [ISO/IEC, 1995] se establece que "el Proceso de Mantenimiento contiene las actividades y tareas realizadas por el mantenedor. Este proceso se activa cuando el producto software sufre modificaciones en el código y la documentación asociada, debido a un problema o a la necesidad de mejora o adaptación. El objetivo es modificar el producto software existente preservando su integridad. Este proceso incluye la migración y retirada del producto software. El proceso termina con la retirada del producto software". (18) (17)

Pressman [1998] dice que "la fase mantenimiento se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que

evoluciona el entorno del software, y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente". (19) (17)

La corrección de defectos en el software, la creación de nuevas funcionalidades en el este debido a nuevos requisitos de usuario y la mejora de funcionalidades o del rendimiento, son las tres actividades fundamentales que se realizan en esta etapa. (20)

En las anteriores definiciones de mantenimiento aparecen indicados, directa o indirectamente, cuatro tipos de mantenimiento: correctivo, adaptativo, perfectivo y preventivo, los cuales serán abordados más adelante.

El mantenimiento enfoca simultáneamente cuatro aspectos mayores de la evolución del sistema: (21)

- ✓ Mantener el control sobre las funciones diarias del sistema.
- ✓ Mantener el control sobre las modificaciones del sistema.
- ✓ Perfeccionar las funciones aceptables existentes.
- ✓ Impedir que el desempeño del sistema se degrade a niveles inaceptables.

Entre sus ventajas se encuentran:

- ✓ Confiabilidad: los equipos operan en mejores condiciones de seguridad, ya que se conoce su estado y sus condiciones de funcionamiento.
- ✓ Mayor duración de los equipos instalados.
- ✓ Uniformidad en la carga de trabajo para el personal del mantenimiento, debido a una programación de actividades.
- ✓ Menor costo de reparaciones.

Sin embargo, es muy probable que se originen algunas fallas al momento de la ejecución, además de que a veces puede ser muy costoso y tampoco se puede asegurar el tiempo que demore ejecutarlo.

1.3.1 Actividades del mantenimiento de software

El desconocimiento de las actividades que implica el mantenimiento del software puede inducir a minusvalorar su importancia, y se tiende a asociar el mantenimiento del software con la corrección de errores en los programas.

Por esta causa, la impresión más generalizada entre los gestores, usuarios, e incluso entre los propios informáticos, es que la mayor parte del mantenimiento que se realiza en el mundo es de tipo perfectivo. (22)

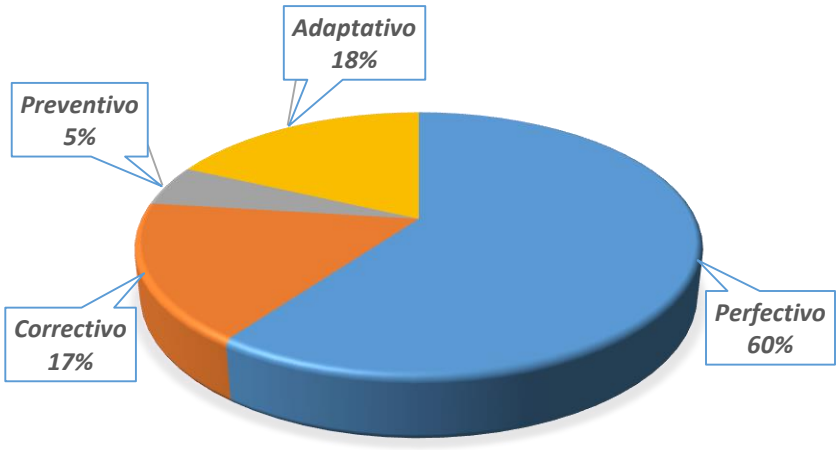

Ilustración 3. Porcentajes dedicados a los diferentes tipos de mantenimiento.

A continuación se explican los diferentes tipos de actividades que se realizan con cada modificación del software: (22)

- ✓ *Análisis de impacto y de costes/beneficios:* se dedica esta actividad a analizar diferentes alternativas de implementación y/o a comprobar su impacto en la planificación, coste y facilidad de operación.
- ✓ *Comprensión del cambio:* puede consistir en localizar el error y determinar su causa, o en comprender los requisitos de una mejora solicitada.
- ✓ *Diseño del cambio:* se refiere al diseño propuesto para el cambio, pudiéndose incluir un rediseño del sistema.
- ✓ *Codificación y pruebas unitarias:* se codifica y prueba el funcionamiento de cada componente modificado.
- ✓ *Inspección, certificación y consultoría:* esta actividad se dedica a inspeccionar el cambio, comprobar otros diseños, reuniones de inspección, etc.
- ✓ *Pruebas de integración:* se refiere a comprobar la integración de los componentes modificados con el resto del sistema.
- ✓ *Pruebas de aceptación:* en esta actividad, el usuario comprueba, junto al personal encargado del mantenimiento, la adecuación del cambio a sus necesidades.
- ✓ *Pruebas de regresión:* en esta actividad se somete el software modificado a casos de pruebas previamente almacenados y por los que ya pasó.
- ✓ *Documentación del sistema:* se revisa y reescribe, en caso necesario, la documentación del sistema para que se ajuste al producto software ya modificado.

- ✓ *Otra documentación (del usuario, por ejemplo):* se revisa y reescribe, en caso necesario, los diferentes manuales de usuario y otra documentación, excepto la documentación del sistema.
- ✓ *Otras actividades*, como las dedicadas a la gestión del proyecto de mantenimiento.

1.3.2 Tipos de mantenimiento

Mantenimiento Correctivo

A pesar de las pruebas y verificaciones que aparecen en el ciclo de vida del software, los programas pueden tener defectos. El mantenimiento correctivo tiene por objetivo localizar y eliminar estos defectos de los programas, que son características del sistema que causan fallos. Un fallo ocurre cuando el comportamiento de un sistema es diferente del establecido en la especificación.

En la siguiente figura se muestra una distribución de las causas de los defectos. (23)

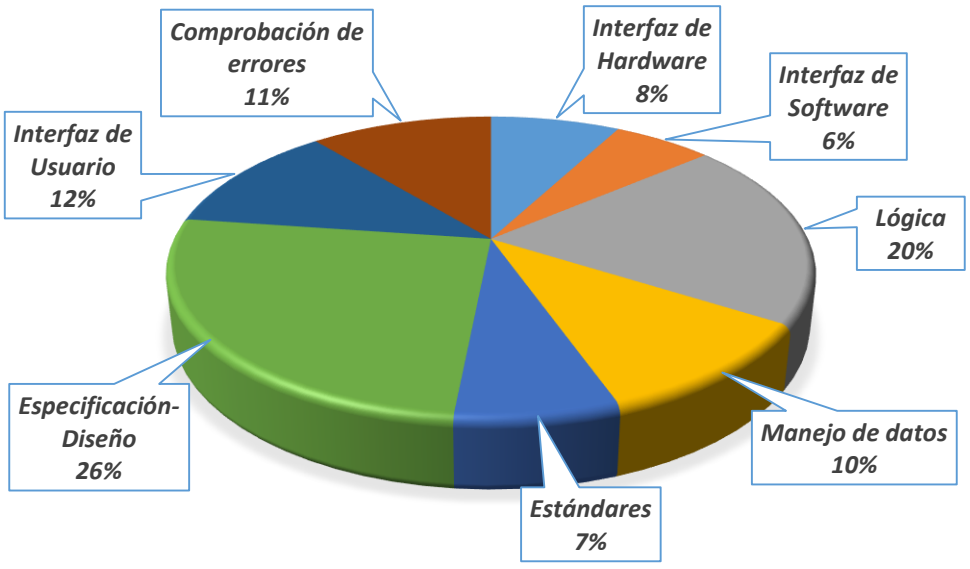

Ilustración 4. Origen de los defectos del software.

Entre otros, los fallos en el software pueden ser de:

- ✓ *Procesamiento:* salidas incorrectas de un programa.
- ✓ *Rendimiento:* tiempo de respuesta demasiado alto en una búsqueda de información.
- ✓ *Programación:* inconsistencias en el diseño de un programa.

- ✓ *Documentación:* inconsistencias entre la funcionalidad de un programa y el manual de usuario.

Mantenimiento Adaptativo

El mantenimiento adaptativo consiste en la modificación de un programa debido a cambios en el entorno (hardware o software) en el cual se ejecuta, los cuales pueden afectar al sistema operativo (cambio a uno más moderno), a la arquitectura física del sistema informático (paso de una arquitectura de red de área local a Internet/Intranet) o al entorno de desarrollo del software (incorporación de nuevos elementos o herramientas como ODBC[1]). (23)

El tipo de cambio necesario puede ser muy diferente: desde un pequeño retoque en la estructura de un módulo hasta tener que reescribir prácticamente todo el programa para su ejecución en un ambiente distribuido en una red.

Los cambios en el entorno software pueden ser de dos clases:

- ✓ *En el entorno de los datos*, por ejemplo, al dejar de trabajar con un sistema de ficheros clásico y sustituirlo por un sistema de gestión de bases de datos relacionales.
- ✓ *En el entorno de los procesos*, por ejemplo, migrando a una nueva plataforma de desarrollo con componentes distribuidos, Java, ActiveX, etc.

Mantenimiento Perfectivo

Cambios en la especificación, normalmente debidos a cambios en los requisitos de un producto software, implican un nuevo tipo de mantenimiento denominado perfectivo. Puede ser desde algo tan simple como cambiar el formato de impresión de un informe, hasta la incorporación de un nuevo módulo aplicativo. Se puede definir el mantenimiento perfectivo como el conjunto de actividades para mejorar o añadir nuevas funcionalidades requeridas por el usuario. (23)

Existen dos tipos de mantenimiento perfectivo:

- ✓ *De Ampliación*: orientado a la incorporación de nuevas funcionalidades.
- ✓ *De Eficiencia*: busca la mejora de la eficiencia de ejecución.

Este tipo de mantenimiento aumenta cuando un producto software tiene éxito comercial y es utilizado por muchos usuarios, ya que cuanto más se utiliza un software, más

[1] Open DataBase Connectivity: estándar de acceso a las bases de datos.

peticiones de los usuarios se reciben demandando nuevas funcionalidades o mejoras en las existentes.

Mantenimiento Preventivo

Este último tipo de mantenimiento consiste en la modificación del software para mejorar sus propiedades (por ejemplo, aumentando su calidad y/o su mantenimiento) sin alterar sus especificaciones funcionales. Por ejemplo, se pueden incluir sentencias que comprueben la validez de los datos de entrada, reestructurar los programas para mejorar su legibilidad, o incluir nuevos comentarios que faciliten la posterior comprensión del programa. (23)

En algunos casos se ha planteado el mantenimiento para la reutilización, consistente en modificar el software (buscando y modificando componentes para incluirlos en bibliotecas) para que sea más fácilmente reutilizable. En realidad este tipo de mantenimiento es preventivo, especializado en mejorar la propiedad de reusabilidad del software.

1.3.3 Selección del tipo de mantenimiento

Luego de analizar el concepto y las características de los diferentes tipos de mantenimiento y enfocado al desarrollo de la investigación, se identificó que el mantenimiento perfectivo y el preventivo son los que están involucrados con la propuesta de solución, ya que esta consiste en un conjunto de sentencias que comprobarán la correspondencia entre los datos de entrada y la respuesta esperada en uno de los módulos del sistema. Además, el principal objetivo de la suite de pruebas es detectar los errores existentes para su posterior corrección por parte del equipo de desarrollo del COJ, es decir, contribuir a mejorar su rendimiento.

1.4 Lenguaje de programación

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas que definen su estructura y el significado de sus elementos y expresiones. (24) Algunos ejemplos son:

- ✓ Ensamblador

- ✓ C++
- ✓ Java
- ✓ C#

1.4.1 Java

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier Máquina Virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. (25)

Se creó con cinco objetivos principales:

1. Usar el paradigma de la programación orientada a objetos.
2. Permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Incluir por defecto soporte para trabajo en red.
4. Diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Su diseño, robustez, respaldo de la industria y fácil portabilidad, han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Algunas de sus aplicaciones se muestran a continuación:

- ✓ Dispositivos móviles y sistemas embebidos.
- ✓ Navegador web.
- ✓ Sistemas de servidor.
- ✓ Aplicaciones de escritorio.

Se decide el uso de Java por las razones siguientes:

- ✓ Es un lenguaje muy utilizado por la UCI al no ser propietario.
- ✓ El COJ está desarrollado en este lenguaje.
- ✓ Los servidores actuales sobre los que está desplegado el COJ son servidores de aplicaciones Java.

- ✓ El ambiente de trabajo utilizado actualmente por el equipo de desarrollo del COJ es Java, otro lenguaje se saldría de los esquemas del proceso integral de desarrollo de la aplicación cliente.

1.5 Frameworks de pruebas automatizadas para el lenguaje de programación Java

Un framework de pruebas unitarias es un paquete de software usado por los desarrolladores, con el propósito de automatizar la creación de unidades de pruebas destinadas a comprobar el funcionamiento interno de un programa. Proveen todos los mecanismos necesarios para ejecutar la lógica específica a la prueba, sin tener que preocuparse por la infraestructura necesaria. (26)

Existen varios frameworks de pruebas disponibles para Java, entre los que se encuentran Mock y JUnit. A continuación se hace referencia a ellos ya que fueron los seleccionados por el equipo de desarrollo para la implementación de la suite de pruebas, teniendo en cuenta las características de estas herramientas, del COJ y de la propuesta de solución.

1.5.1 Mock

Un mock es un tipo concreto de doble de test. La expresión “doble” se usa en el mismo sentido de los actores “dobles” en las películas de acción, ya que se hace pasar por un colaborador del SUT[2] cuando en realidad no es la entidad que dice ser. Constituye un objeto preprogramado con expectativas que conforman la especificación de cómo se espera que se reciban las llamadas. Son más complejos que los stubs[3] aunque sus diferencias son sutiles. Estos últimos proporcionan respuestas predefinidas a llamadas hechas durante los tests, frecuentemente, sin responder en absoluto a cualquier otra cosa fuera de aquello para lo que ha sido programado. (27)

Está basado en EasyMock[4], y el funcionamiento es prácticamente parecido, aunque mejora la API (Interfaz de Programación de Aplicaciones) a nivel sintáctico, haciéndolo

[2] SUT (Subject under test): objeto que nos ocupa, el que estamos diseñando a través ejemplos

[3] Stubs: código usado como sustituto de alguna otra funcionalidad
[4] EasyMock: primer generador de objetos Mock dinámico.

más entendible para todos y además permite crear mocks de clases concretas (y no sólo de interfaces). (28)

La implementación de tests Mockito se hace en varias etapas:

- ✓ Creación del objeto mock.
- ✓ Descripción del comportamiento esperado.
- ✓ Uso de los mocks.
- ✓ Verificación que la interacción con los mocks sea correcta.

Los mocks presentan dos inconvenientes fundamentales:

- ✓ El código del test puede llegar a ser difícil de leer.
- ✓ El test corre el riesgo de volverse frágil si conoce demasiado bien el interior del SUT.

1.5.2 JUnit

JUnit es un conjunto de bibliotecas que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

El propio framework incluye formas de ver los resultados (runners) que pueden ser en modo texto o gráfico. Este tipo de herramientas hacen que el esfuerzo y el trabajo en la fase de pruebas se reduzcan, permitiendo que el desarrollador o tester se centre en la verificación de resultados correctos y no escribiendo código extenso para realizar sus pruebas.

Integraciones de jUnit

La inmensa mayoría de los IDE de desarrollo se integran con jUnit facilitando el trabajo con el framework. Ejemplo de los mismos son:

- ✓ Eclipse
- ✓ Netbeans
- ✓ Intellij IDEA

1.6 Herramientas de pruebas automatizadas

La realización de testeos de calidad, bien preparados y diseñados, es un factor clave para el óptimo desarrollo de una aplicación informática. Sin testeos bien detallados y planificados una aplicación puede llegar al cliente con multitud de fallos, algo que es inaceptable en un proyecto. Por ello, es importante identificar una serie de herramientas que ayuden a realizar esta labor tan importante. (29)

Posteriormente se detallarán algunas de ellas, explicando las características fundamentales de cada una.

1.6.1 Selenium IDE

Es un plugin de Mozilla Firefox que pertenece al juego de herramientas SeleniumHQ, permite realizar juegos de pruebas sobre aplicaciones web. Para ello realiza la grabación de la acción seleccionada (navegación por una página) en un "script", el cual se puede editar y parametrizar para adaptarse a los diferentes casos, y lo que es más importante su ejecución se puede repetir tantas veces como se quiera. El principal objetivo de este plugin es crear pruebas funcionales, aunque no se puede pasar por alto que este tipo de herramientas permite automatizar tareas que requieren un cierto procesamiento mental básico:

- ✓ Rellenar formularios (autenticación o cualquier otro tipo).
- ✓ Navegación web.
- ✓ Acciones de gestión (CRUD de comentarios blog / correos / noticias / etc.).

Entre sus principales características se pueden nombrar: (30)

- ✓ Facilidad de registro y ejecución de los test.
- ✓ Autocompletado para todos los comandos.
- ✓ Las acciones pueden ser ejecutadas paso a paso.
- ✓ Los test pueden ser almacenados como HTML y scripts Ruby, entre otros formatos.
- ✓ Soporte para Selenium user-extensions.js.
- ✓ Ejecución en varios navegadores.

- ✓ Uso de diferentes API's[5] (Application Programming Interface) en diferentes lenguajes (PHP, Ruby, Java, Javascript, entre otros).

1.6.2 Solex

Es una herramienta Open Source para testeo creado y se implanta como un plugin en Eclipse. Solex permite grabar la sesión de un usuario, permitiéndolo configurar en función a diferentes parámetros, para poder ser utilizado posteriormente y ser repetida, de manera que aseguremos la no regresión de la aplicación. Asimismo, también nos permite realizar pruebas de estrés y rendimiento contra la aplicación web. (29)

1.6.3 WebCorder

Se trata de una herramienta de prueba de software de interfaz gráfica de usuario libre, desarrollado para permitir una sencilla prueba web del usuario final. En esencia, el usuario pulsa grabar y navega su camino a través de un escenario, diciéndole al programa para comprobar si el texto / imágenes en el camino y, opcionalmente, realizar capturas de pantalla. Al final del proceso de detener la grabación y guardar el guión. A continuación, puede reproducir la secuencia de comandos de forma interactiva o en modo por lotes, y el programa va a generar los archivos de registro, etc. (31)

1.6.4 Wireshark

Es probablemente uno de los mejores analizadores de tráfico de red. Más aún si se tiene en cuenta su relación calidad/precio. Es una de las herramientas básicas que se usan en auditorías de seguridad y test de intrusión. Además, sirve en muchas otras pruebas, o incluso en el desarrollo de software, cuando se debe verificar qué ocurre con una aplicación, y si está funcionando correctamente. El objetivo principal de la herramienta es mostrar al usuario todo lo que está circulando a través de su tarjeta de red, conocer que está circulando en el mundo al que estamos conectados. (32)

Algunas características principales son:

- ✓ Funciona bajo varias plataformas como Windows, Linux o Mac OS.
- ✓ Captura de paquetes on the fly, es decir, en tiempo real.

[5] API: conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas.

- ✓ Información detallada de los paquetes. La gestión de los paquetes se realiza bajo extensiones CAP, PCAP, etcétera.
- ✓ Control de sniffing remoto. Esta característica no es muy conocida por muchos de los usuarios de la herramienta, pero se puede colocar un agente en un equipo y dejar escuchando los paquetes que circulan por dicho equipo y reenviarlos, a modo de espejo, a un equipo dónde se encuentre el auditor.
- ✓ Importación y exportación de paquetes.

1.6.5 WebInject

WebInject es una herramienta gratuita para pruebas automatizadas de aplicaciones y servicios web. Puede ser utilizado para probar los componentes individuales del sistema que tienen interfaces HTTP (JSP, ASP, CGI, PHP, AJAX, servlets, formularios HTML, XML / SOAP Web Services, el descanso, etc.), y se pueden utilizar como un instrumento de prueba para crear un conjunto automatizado de pruebas funcionales, de aceptación y de regresión. Constituye un instrumento de pruebas que permite ejecutar muchos casos de prueba y reportar sus resultados. WebInject ofrece resultados en tiempo real y también puede ser usada para monitorear los tiempos de respuesta del sistema. Opcionalmente, se puede utilizar como un corredor de prueba independiente (texto de la solicitud / consola) que puede ser integrado y llama de otros marcos de prueba o aplicaciones. (33)

1.7 Entorno de desarrollo integrado

Un entorno de desarrollo integrado (IDE por sus siglas en inglés) es una herramienta informática que aporta funcionalidades al desarrollador durante todas las etapas del ciclo de vida del desarrollo de software, desde el análisis y diseño a la distribución del producto y su mantenimiento.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto. (34)

1.7.1 IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo Java creado por Jet Brains del que existen dos distribuciones: Comunity Edition (open source) y Ultimate (comercial). Sus creadores definen este IDE como el más inteligente del mundo. Cada aspecto de IntelliJ IDEA está diseñado específicamente para maximizar la productividad del desarrollador. (35)

Soporta tecnologías como:

- ✓ Groovy
- ✓ Android
- ✓ Javascript
- ✓ Struts
- ✓ Spring
- ✓ Hibernate
- ✓ JSF

Para el desarrollo de la solución propuesta se seleccionó IntelliJ IDEA, debido a que brinda los siguientes beneficios:

- ✓ Autocompletado de código.
- ✓ Viene con todo instalado de serie. Además existe un amplio set de plugins.
- ✓ Sensación de fiabilidad y robustez muy superior a otros entornos.
- ✓ Herramienta de refactorización extremadamente inteligente.
- ✓ Ofrece soporte a varios lenguajes de programación (entre ellos Java), a herramientas de control de versiones y al potente framework de pruebas JUnit.

Se decide el uso de IntelliJ IDEA por sus características y por las razones siguientes:

- ✓ Soporta el lenguaje de programación Java.
- ✓ El equipo de desarrollo del COJ utiliza este IDE.
- ✓ Permite efectuar pruebas individuales sin tener que ejecutar las restantes.

1.8 Conclusiones del capítulo

En el presente capítulo se abordaron características de las de pruebas de software, proporcionando los elementos teóricos necesarios para guiar el proceso de definición de una estrategia, orientada a la creación de una suite de pruebas automatizadas para el COJ. También se estudiaron algunas herramientas, seleccionando aquellas que por sus características se adecuan mejor a la solución propuesta, en este caso se

determinaron las siguientes: Mock, JUnit e IntelliJ IDEA. Se identificó además que los tipos de mantenimiento que guiarán el desarrollo de la solución, son el perfectivo y el preventivo, conclusión a la que fue posible arribar teniendo en cuenta las particularidades de los mismos y las necesidades del sistema.

Capítulo 2: Descripción de la propuesta de solución

Debido a la existencia de un gran número de herramientas de pruebas automáticas, se hace necesario investigar cuáles de ellas se ajustan a las características del proyecto COJ. En el presente capítulo se describirán los aspectos que se tuvieron en cuenta para la selección de las herramientas de prueba, se realizará una descripción de la propuesta de solución y para un mayor entendimiento de la misma, se detalla mediante el diagrama de paquetes cómo está estructurado el módulo Administración del COJ.

2.1 Utilización de herramientas de pruebas automatizadas

En la actualidad existe una gran variedad de herramientas para realizar pruebas automatizadas. Estas proporcionan una ventaja evidente en cuanto a las pruebas manuales, sobre todo en el tiempo y la conservación de los recursos, sin comprometer la calidad del procedimiento de prueba, la exactitud de los informes finales y la eficiencia del proceso; son efectivas y económicamente factibles. El uso de las mismas permite mayor facilidad en la detección de los errores y fallos que se producen antes de que el producto de software es liberado, o en momentos en que hay una necesidad de proporcionar asistencia al cliente para resolver los defectos encontrados. (36)

Con el objetivo de evaluar el nivel de conocimiento existente en cuanto a pruebas automatizadas de software y las herramientas que se utilizan en ellas, se realizó una encuesta en dos áreas de la universidad vinculadas a la producción y/o evaluación de software, la cual estuvo compuesta por seis preguntas previamente elaboradas (Anexo # 1).

En las dos primeras se aborda el tema de las pruebas automatizadas de software que se aplican o no en un área determinada. Las restantes cuatro preguntas se enfocan a las herramientas de prueba funcionales, para de esta forma conocer cuáles se utilizan, si se apoya el uso de las mismas y las posibilidades que brinda su uso. Se realizó en dos áreas escogidas, Calisoft y el Grupo de Calidad del CEIGE para un total de 16 encuestados.

Se utilizó el Microsoft Word como herramienta para generar las gráficas correspondientes a la información recogida en dicha encuesta (Anexo 2). A continuación se muestran algunos de los datos obtenidos.

En la ilustración 5 se puede observar que el grado de conocimiento de los encuestados sobre las pruebas de software es de un 100%.

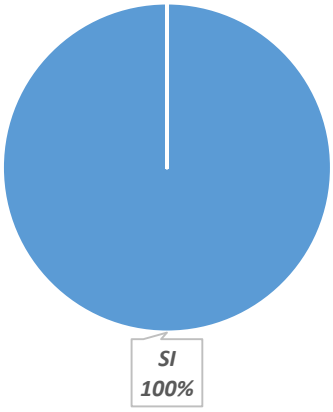

Ilustración 5. Conocimiento de pruebas automatizadas de software.

En la ilustración 6 correspondiente a la pregunta 1.1, se específica hasta qué punto cada uno de los 16 encuestados tiene conocimientos acerca de una determinada prueba, por ejemplo en el caso de las pruebas funcionales el 100 % las conoce.

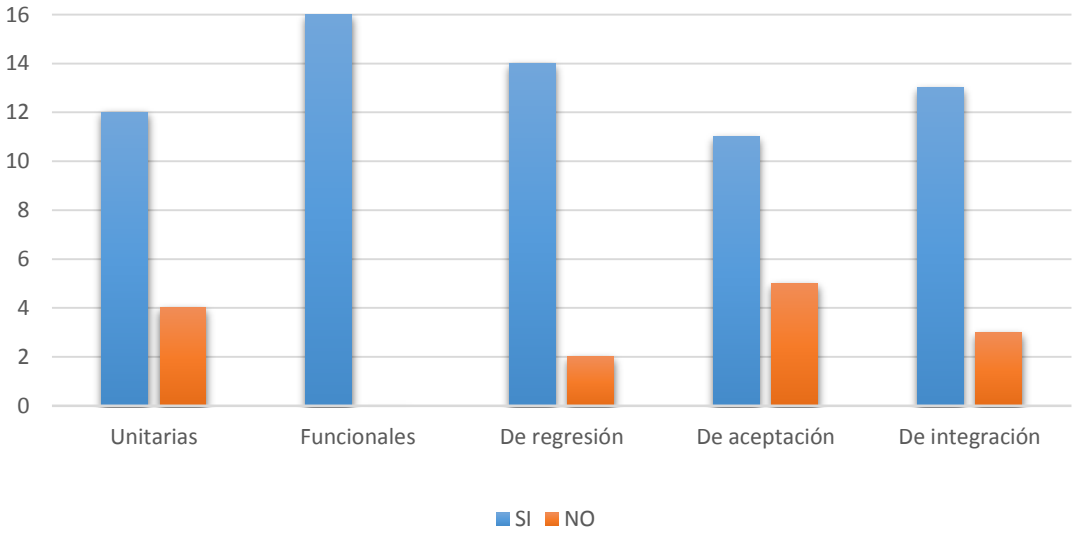

Ilustración 6. Conocimiento de pruebas automatizadas específicas.

En la siguiente ilustración (pregunta 2.1) se evidencia que las pruebas funcionales son las más aplicadas en ambos centros.

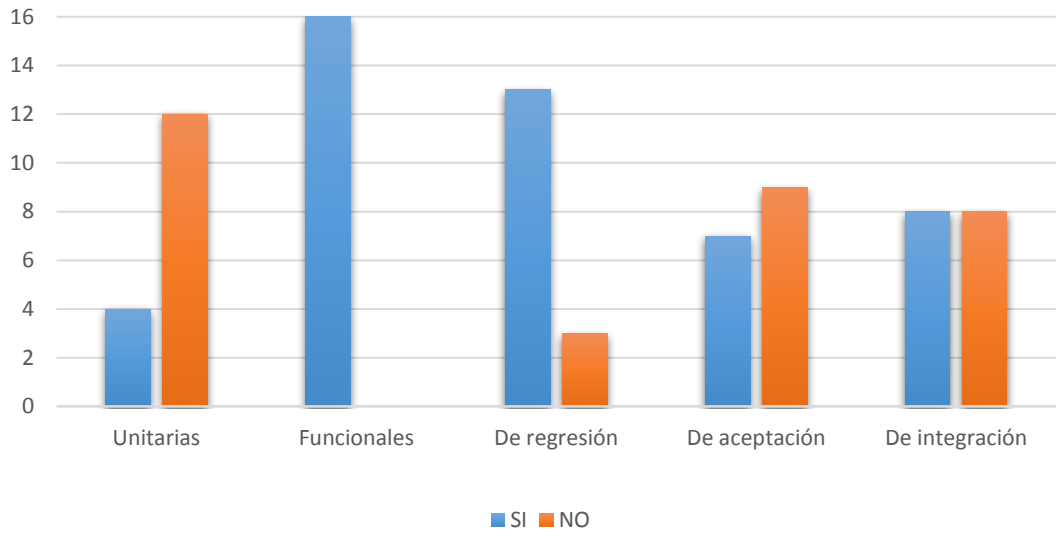

Ilustración 7. Aplicación de pruebas automatizadas específicas.

La ilustración 8 muestra los resultados de la pregunta 5, donde se puede observar el dominio de los encuestados de cada una de las herramientas especificadas, dándose el caso de un nulo conocimiento acerca de Solex, Imprimatur, Capedit y WebInject.

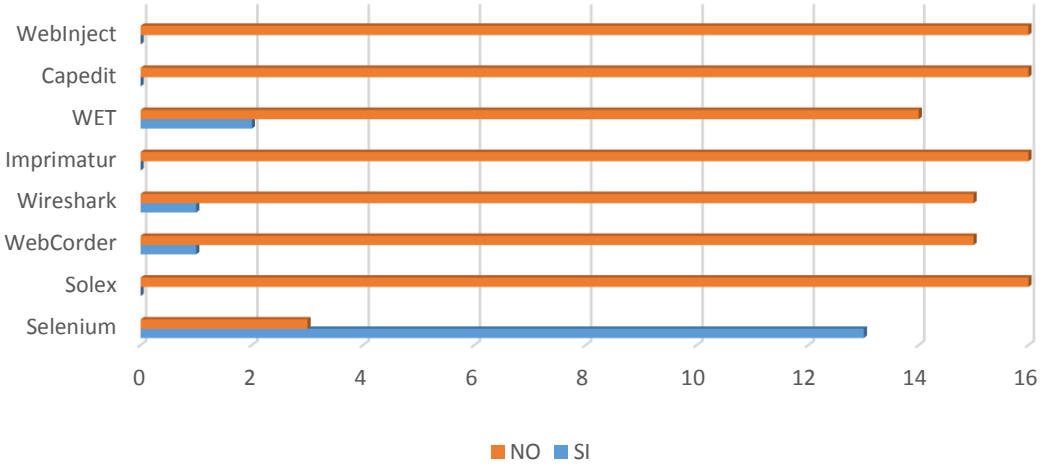

Ilustración 8. Conocimiento de herramientas específicas.

La ilustración 9 de la pregunta 6 muestra la opinión de los encuestados sobre el uso de las pruebas funcionales.

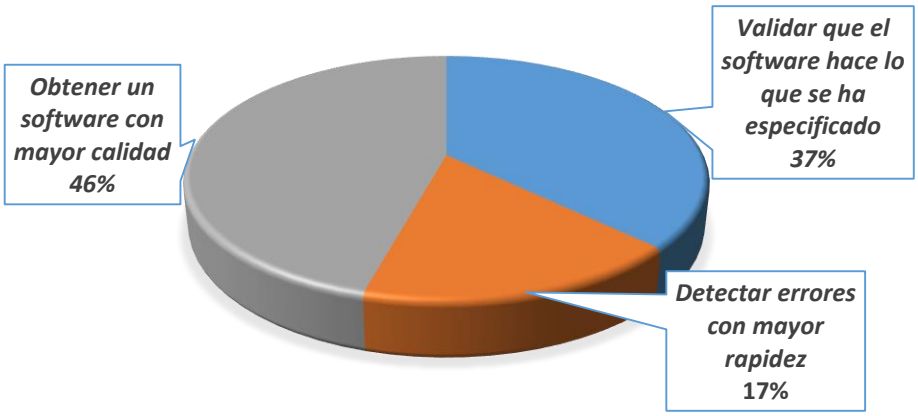

Ilustración 9. Importancia de la realización de pruebas funcionales.

2.1.1 Resultados de la encuesta

Luego de la aplicación de la encuesta se pudo arribar a los siguientes resultados:

- ✓ Existe un conocimiento de las pruebas automatizadas de software, pero no se ha ampliado el espectro hacia la variedad de las mismas, sus usos y potencialidades, lo que conlleva a desecharlas en muchas ocasiones.
- ✓ A pesar de estar al tanto del proceso de evaluación de software, algunas de las pruebas que se emplean no se aplican en su totalidad, sólo se hace uso de las más generales dejando a un lado otras también útiles para el sistema.
- ✓ Las herramientas de prueba son desconocidas por gran parte de los encuestados, aunque esto ocurra se aboga por su uso en gran medida sabiendo que facilitarían exponencialmente el trabajo.
- ✓ La aplicación de la encuesta sirvió de guía al equipo de desarrollo para determinar qué herramienta sería más factible utilizar, en este caso se concluyó que Selenium IDE, ya que al brindar facilidades de registro y ejecución de los test, permite una mejor interacción del usuario con los mismos. Además al ser la más utilizada en la universidad, resulta más sencilla la obtención de documentación relacionada con la herramienta y la consulta a personas con vastos conocimientos acerca de ella, por lo que es recomendable su uso.

2.2 Descripción de la propuesta de solución

La propuesta de solución consiste en desarrollar una suite de pruebas automatizadas para el COJ, con el objetivo de comprobar el comportamiento de las funcionalidades del módulo Administración donde además se encuentran otros elementos, verificando que todos estos funcionen según las restricciones descritas en el validador de cada campo. Para lograrlo se utilizará JUnit en la creación de pruebas java de forma controlada, permitiendo generar código en menor cantidad de tiempo. Con el uso de Mockito se pretende simular el comportamiento complejo de los objetos, además de facilitar el desarrollo de test unitarios y la detección de errores.

Para un mayor entendimiento se exponen a continuación las características del proceso de pruebas del COJ que actualmente se realiza de forma manual y posteriormente, una breve descripción luego de integrada la suite al sistema.

2.2.1 Proceso de pruebas manuales

Las pruebas manuales de software en el COJ son realizadas por una o varias personas capacitadas para desarrollarlas, que interactúan en la computadora con la interfaz de la aplicación. Durante este proceso los probadores verifican mediante el uso de diversas combinaciones de entrada, que los resultados obtenidos estén en correspondencia con el comportamiento esperado y además, realizan el registro de sus observaciones.

Este tipo de pruebas se repiten con frecuencia durante todo el ciclo de vida del sistema, ya que el mismo está sujeto a constantes cambios, principalmente en cuanto a las funcionalidades que ofrece y a su interfaz.

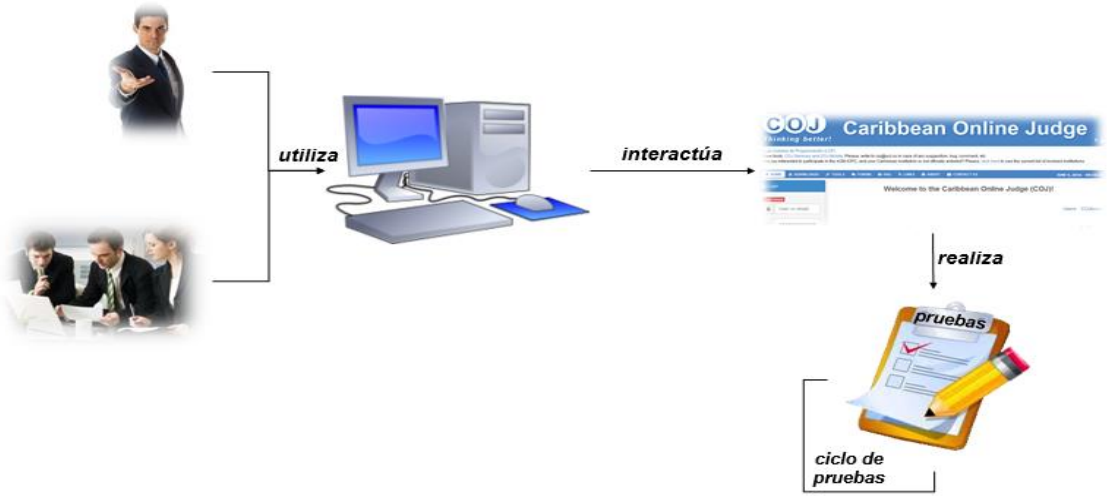

Ilustración 10. Proceso de pruebas manuales.

2.2.2 Proceso de pruebas automatizadas

Con la integración de la suite de pruebas automatizadas al COJ, ocurre un cambio considerable respecto a la calidad del proceso de pruebas que se realiza en la actualidad, brindando gran facilidad y rapidez en la detección de defectos. También permite medir el grado con que el sistema cumple los requerimientos descritos por el usuario. Este tipo de prueba constituye un mecanismo para comprobar de forma automática el correcto funcionamiento del COJ, mostrando todos los errores que se detecten.

En este caso no serán necesarias varias personas para llevar a cabo las pruebas. De igual forma el responsable del proceso debe interactuar con la aplicación y luego ejecutar la suite, la cual se encarga de realizar la detección de errores de forma automática. Lo mismo sucediera si se realiza una modificación en el sistema.

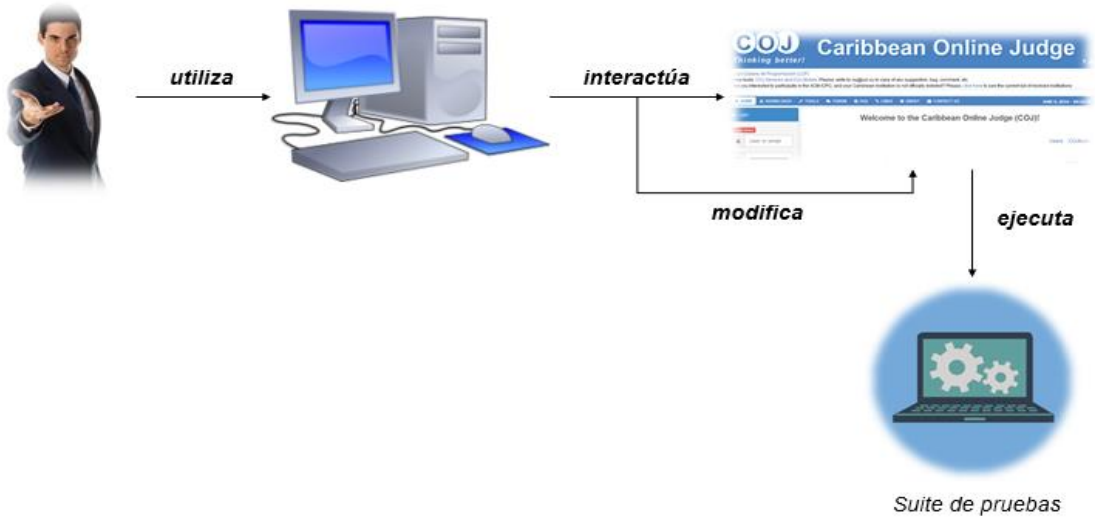

Ilustración 11. Proceso de pruebas automatizadas.

2.2.3 Diagrama de paquetes

Un diagrama de paquetes en el Lenguaje Unificado de Modelado representa las dependencias entre los paquetes que componen un modelo. Es decir, suministra una descomposición de la jerarquía lógica del sistema, su división en agrupaciones y las dependencias entre ellas. (37)

A continuación se muestra el diagrama correspondiente al módulo Administración del COJ, donde los paquetes están organizados con el objetivo de maximizar la comprensión de la estructura del sistema. El elemento básico que se evidencia es el paquete (visualmente está representado como una carpeta), con el propósito de agrupar

elementos relacionados semánticamente. Proporciona un espacio de nombres encapsulado dentro del cual todos ellos son únicos.

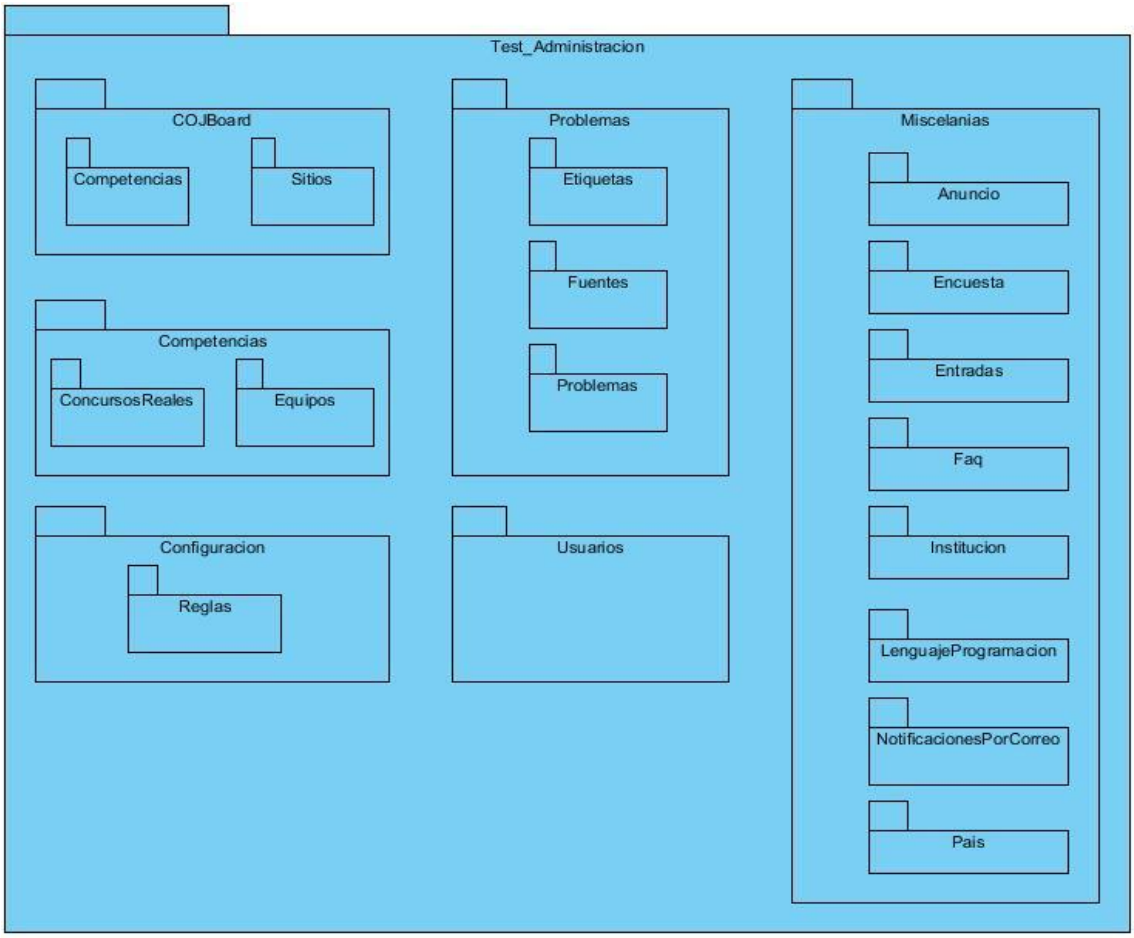

Ilustración 12. Diagrama de paquetes del módulo Administración del COJ.

2.3 Aplicación de las herramientas de pruebas seleccionadas en la suite

En la selección de las herramientas se hizo énfasis en las libres y de código abierto por el ahorro económico que representan y las posibilidades de personalización a los intereses tecnológicos del centro. También se enfocaron al entorno donde serán utilizadas, y a las características del producto al cual se le aplicarán estas pruebas automatizadas. Una mala elección además de gastar tiempo y recursos, puede incidir en el avance del proceso de prueba y generar un obstáculo difícil de resolver.

2.3.1 JUnit

Este framework facilita la creación de código para la automatización de pruebas con presentación de los resultados. Con él se verifica si cada método de las clases que se encuentran en el módulo Administración funciona de la forma esperada, mostrando posibles errores o fallos.

Algunas ventajas de su uso para el desarrollo de la suite son:

- ✓ Permite la creación rápida de código de prueba, contribuyendo de esta forma al ahorro en términos de tiempo.
- ✓ Una vez escritas las pruebas son ejecutadas rápidamente sin que, para eso, sea interrumpido el proceso de desarrollo del sistema.
- ✓ JUnit comprueba los resultados de las pruebas realizadas y suministra una respuesta inmediata.
- ✓ Permite crear una jerarquía de pruebas para probar sólo un paquete del módulo o el módulo en su totalidad.
- ✓ La selección de JUnit permite que se pierda menos tiempo depurando el código.

2.3.2 Mock

Mockito es un framework para pruebas de aplicaciones Java, que resulta útil en la investigación porque:

- ✓ Permite simular comportamientos complejos de objetos.
- ✓ Brinda facilidades a los usuarios en la detección de errores.
- ✓ Reduce drásticamente el número de líneas de código, permitiendo de esta forma la concisión del mismo.

La idea de las pruebas al usar Mockito es el concepto de stubbing – ejecutar – verificar (programar un comportamiento, ejecutar las llamadas y verificar las llamadas). (38)

2.3.3 Selenium IDE

Selenium facilitará escribir test con las interacciones del usuario y ejecutarlos directamente desde el navegador. Es compatible con el lenguaje de programación Java, que fue el seleccionado para el desarrollo de la suite de pruebas.

De forma general, su selección está basada en los siguientes aspectos:

- ✓ Uno de los resultados arrojados por la encuesta aplicada para medir el nivel de conocimiento que existe en cuanto al proceso de pruebas de software y las herramientas de pruebas automatizadas que se utilizan, fue que Selenium es la más conocida en los centros donde se aplicó, por lo que sería factible su uso por las razones antes explicadas en los resultados de la encuesta.
- ✓ Permite la ejecución de tests funcionales sobre una página web.
- ✓ El equipo de desarrollo puede crear sus propios tests y usarlos posteriormente, mediante varios comandos o funciones, con una serie de parámetros que dicha herramienta ofrece.
- ✓ Brinda facilidad en la verificación de elementos en la página, debido a que se puede comprobar si un elemento está presente en un lugar específico de la página o en cualquier otro sitio.

2.4 Escenarios de prueba

Selenium

Descripción del escenario para pruebas funcionales con Selenium IDE	
Precondiciones	-Visualizar Selenium IDE. -Debe escoger la aplicación a la cual se le realizará la prueba así como su dirección URL. -Definir un escenario de prueba, ejemplo: Registrar datos. -Definir un nuevo caso de prueba grabando una secuencia de acciones sobre la aplicación. -Salvar la grabación realizada. -Ejecutar la prueba. -Probar nuevos casos de pruebas hechos por el usuario.
Entrada	-Se selecciona manualmente la opción de grabar en la herramienta Selenium IDE.
Procedimientos	-La herramienta registra cada página del sistema a la cual se accedió durante la grabación así como sus aspectos individuales.
Poscondiciones	-La herramienta grabará cada paso que realice el usuario dentro del sistema.

Salida	-La herramienta genera una tabla con un formulario con tres campos: 'Command', 'Target' y 'Value'. Mediante estos tres campos es posible agregar comandos adicionales a la tabla de ejecución o editar los existentes.
Análisis de resultados	-Muestra si la aplicación está trabajando con todas sus funcionalidades de forma correcta.

Tabla 1. Escenario de prueba con Selenium.

JUnit

Descripción del escenario para pruebas funcionales con JUnit	
Precondiciones	-Se escoge la funcionalidad a probar. -Se genera una clase automáticamente que implementa la clase a probar.
Entrada	-Se declaran manualmente las pruebas a realizar.
Procedimientos	-Ejecuta la prueba definida anteriormente. -Comprueba automáticamente que no existen errores en la definición de la prueba.
Poscondiciones	-No presenta
Salida	-Devuelve si se ejecutó correctamente la prueba. -Brinda estadísticas.
Análisis de resultados	-Permite determinar si una funcionalidad devuelve el resultado esperado. -Muestra si presenta algún error que dificulte el correcto funcionamiento del sistema. -Permite verificar la apropiada aceptación de los datos.

Tabla 2. Escenario de prueba con JUnit.

Luego de un análisis exhaustivo de las herramientas comentadas en el capítulo 1, se realizó una selección de aquellas que cumplieran con las características del centro donde se desean aplicar y se logró como resultado que JUnit, Mock y Selenium son las más propicias para conformar la suite de pruebas.

2.5 Selección de las métricas de mantenimiento

Las métricas permiten saber, entre otras cosas, el número o importancia de los errores que se detectan en los tests o correspondientes a reclamaciones recibidas por parte del cliente. Todas las métricas de software pueden usarse para el desarrollo de uno nuevo y para el mantenimiento del existente. (39)

Para comprobar que la propuesta de solución contribuye al mantenimiento del COJ, se definieron una serie de criterios que deben ser valorados por varias personas familiarizadas con el proceso de pruebas del COJ y con la suite a desarrollar:

Criterios de completitud:

- ✓ La propuesta está correcta.
- ✓ Cumple con las funcionalidades requeridas.
- ✓ Muestra los resultados deseados.

Criterios de implantación:

- ✓ Necesidad del empleo de la propuesta.
- ✓ Posibilidades de aplicación.

Criterios de flexibilidad:

- ✓ Capacidad de la propuesta para la admisión de cambios.
- ✓ Adaptabilidad a otros módulos del sistema.

Criterios de impacto:

- ✓ Impacto en el sistema para el cual está destinado la propuesta.
- ✓ Optimización del proceso de pruebas.

Criterios de usabilidad:

- ✓ La propuesta es de fácil entendimiento.
- ✓ No se requiere conocimientos avanzados por los usuarios.

2.6 Conclusiones del capítulo

Durante el desarrollo del presente capítulo se aplicó una encuesta a miembros de los grupos de calidad en Calisoft y en el CEIGE, la cual sirvió como apoyo en la selección de Selenium IDE para facilitar el registro y ejecución de los test. Se describieron las herramientas JUnit, Selenium IDE y Mockito, que son las que contribuirán a la implementación de las pruebas automatizadas. Además se describió cómo se realiza el

proceso de pruebas en dos etapas: primero con las pruebas manuales que se ejecutan en el sistema y después con la integración de la suite. Se elaboró un diagrama de paquetes correspondiente al módulo Administración del COJ para un mejor entendimiento de la propuesta de solución, donde se evidencian cada uno de sus componentes y las clases que la conforman. También se presentaron los criterios que servirán de guía en la posterior evaluación del sistema.

Capítulo 3: Implementación y validación

En los capítulos anteriores se evidenció con claridad la necesidad de la utilización de herramientas para la realización de pruebas automáticas. En el actual se describirá cómo se ha aplicado el proceso y la utilización de las herramientas propuestas anteriormente, se analizarán los resultados obtenidos durante la aplicación de las pruebas al Juez el Línea Caribeño, así como el resultado del proceso de validación realizado mediante la técnica de Iadov.

3.1 Implementación

La implementación constituye una de las fases más importantes del desarrollo de software. En ella se toman como punto de partida los resultados obtenidos en el diseño, implementándose el sistema en términos de componentes como ficheros de código binario, código fuente, scripts y ejecutables. Su importancia reside en que se obtiene como consecuencia un sistema ejecutable, siendo esto uno de los principales objetivos en el desarrollo de software. (40)

3.1.1 Estándar de codificación empleado

Con el objetivo de lograr una estandarización en la programación de la suite y facilitar la lectura, comprensión y mantenimiento del código, se decide aplicar el estilo de escritura empleado en estándares de codificación *CamelCase*, específicamente la variante *lowerCamel Case*.

Posteriormente se describen a grandes rasgos las convenciones de nomenclatura.

Clases:

1. Se exceptúan el uso de las tildes y la letra ñ, la que será sustituida por nn.

```
public void setUp() throws Exception {
    MockitoAnnotations.initMocks(this);
    utiles_testAdministracionEtiquetas = new Utiles_TestAdministracionEtiquetas();
    this.mockMvc = MockMvcBuilders.standaloneSetup(classificationController).build();
```

Ilustración 13. Estándar de codificación. Ejemplo 1.

2. El nombre de todas las variables y métodos comenzará con letra minúscula y si este está compuesto por varias palabras, todas las palabras internas que lo

componen comienzan también con minúscula y están separadas por un "_". Ejemplo: validator_email.

```
//Validando nombre
@Test
public void validarNombreEtiqueta() throws Exception {

    ArrayList<String> lista1 = utiles_testAdministracionEtiquetas.getNombreCorrectosEtiqueta();
    ArrayList<String> lista2 = utiles_testAdministracionEtiquetas.getNombreIncorrectosEtiqueta();
```

Ilustración 14. Estándar de codificación. Ejemplo 2.

3. Identación: El contenido siempre se identará con tabs, nunca utilizando espacios en blanco.

```
String direccionAux="/admin/addclassifications.xhtml";
 for (String a : lista1) {

    String newname = a.replace(" ", "");
    if (newname.length() == 0)  Assert.assertEquals("Validacion de nombre incorrecta", "/admin/manageclassifications.xhtml",null );

    r = mockMvc.perform(post(direccionAux)
            .param("name", a)
            .session((MockHttpSession) result.getRequest().getSession()))
            .andReturn();
```

Ilustración 15. Estándar de codificación. Ejemplo 3.

4. Clases: El nombre de las clases comenzará con mayúscula. Si este está compuesto por varias palabras, todas comienzan con mayúscula. Ejemplo: "AAMySuiteTest".
5. Intentar mantener los nombres de las clases descriptivos y simples. Usar palabras completas, evitar acrónimos y abreviaturas, a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL.

```
public class NombreEtiqueta_Test {

    private MockMvc mockMvc;


    @Mock
    private ProblemDAO problemDAO;

    @InjectMocks
    private ClassificationController classificationController;
```

Ilustración 16. Estándar de codificación. Ejemplo 4.

Nombre de variables:

- ✓ No se utilizarán nombres de variables que puedan ser ambiguos.
- ✓ Se procurará evitar dar nombres sin sentido a variables temporales. Por ejemplo: temp, i.

```
@Before
public void setUp() throws Exception {
    MockitoAnnotations.initMocks(this);
    utiles_testAdministracionEtiquetas = new Utiles_TestAdministracionEtiquetas();
    this.mockMvc = MockMvcBuilders.standaloneSetup(classificationController).build();
```

Ilustración 17. Estándar de codificación. Ejemplo 5.

3.1.2 Code Coverage

La cobertura de código es una métrica que permite conocer la cantidad de código (medida porcentualmente) que está siendo cubierto por las pruebas. O sea, al ejecutarlas en la aplicación y si hay alguna línea de código que nunca fue ejecutada en el contexto de las pruebas, entonces dicha línea no está cubierta. En otro caso si el código consta de 100 líneas y solo 50 están siendo ejecutadas al correr las pruebas, entonces la cobertura es del 50%. (41)

La utilización del code coverage aumenta la calidad de la suite debido a que si se tiene una alta cobertura, significa que gran parte del código está siendo probado y por consiguiente se podría tener cierta certeza sobre el correcto funcionamiento de la suite. Al mismo tiempo medir la cobertura ayuda a detectar código innecesario, ya que no se ejecuta. Una cobertura del 100% solo dice que todo el código está siendo cubierto por pruebas, pero puede que las pruebas no estén contempladas algunas situaciones, o sea, que falten algunas. (42)

Con la utilización de esta medida, es posible arribar a las siguientes conclusiones:

1. Es necesaria la realización de más pruebas.
2. Se ha creado código que nunca se va a ejecutar, por lo tanto no es necesario y puede ser eliminado.

A continuación se muestran las medidas ofrecidas por el code coverage para la suite de pruebas automatizadas para el COJ.

97% classes, 86% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
Test_Administracion	0% (0/1)	0% (0/0)	0% (0/1)
Test_Administracion.COJBoard	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.COJBoard.Competencia	100% (6/6)	64% (36/56)	85% (273/321)
Test_Administracion.COJBoard.Sitio	100% (6/6)	62% (35/56)	82% (285/347)
Test_Administracion.Competencias	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.Competencias.ConcursosReales	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.Competencias.ConcursosReales.AdicionarConcurso	100% (1/1)	100% (3/3)	100% (39/39)
Test_Administracion.Competencias.ConcursosReales.ConfiguracionGeneral	100% (6/6)	63% (40/63)	93% (627/674)
Test_Administracion.Competencias.ConcursosReales.ConfiguracionGlobal	100% (1/1)	100% (6/6)	92% (243/263)
Test_Administracion.Competencias.Equipo	100% (8/8)	60% (45/74)	89% (451/506)
Test_Administracion.Configuracion	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.Configuracion.Reglas	100% (2/2)	41% (14/34)	61% (51/83)
Test_Administracion.Miscelaneas	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.Miscelaneas.Anuncio	100% (2/2)	61% (11/18)	62% (44/70)
Test_Administracion.Miscelaneas.Encuesta	100% (5/5)	63% (31/49)	86% (226/262)
Test_Administracion.Miscelaneas.Entradas	0% (0/1)	0% (0/4)	0% (0/67)
Test_Administracion.Miscelaneas.Faq	100% (3/3)	64% (16/25)	83% (94/112)
Test_Administracion.Miscelaneas.Institucion	100% (5/5)	56% (29/51)	79% (226/283)
Test_Administracion.Miscelaneas.LenguajeProgramacion	100% (5/5)	64% (29/45)	86% (206/238)
Test_Administracion.Miscelaneas.NotificacionesPorCorreo	100% (3/3)	65% (15/23)	82% (87/105)
Test_Administracion.Miscelaneas.Pais	100% (5/5)	58% (27/46)	85% (217/254)
Test_Administracion.Problemas	0% (0/0)	0% (0/0)	0% (0/0)
Test_Administracion.Problemas.Etiquetas	100% (2/2)	66% (8/12)	85% (48/56)
Test_Administracion.Problemas.Fuentes	100% (3/3)	55% (16/29)	80% (98/122)
Test_Administracion.Problemas.Problemas	100% (1/1)	100% (4/4)	80% (25/31)
Test_Administracion.Usuarios	100% (1/1)	100% (16/16)	93% (853/914)

Ilustración 18. Medidas ofrecidas por el code coverage.

3.2 Validación de los resultados

Luego del diseño e implementación de las pruebas es necesario demostrar que la solución contribuye al mantenimiento del COJ, teniendo en cuenta su importancia y otras características. De esta forma, se comprueba que la suite constituye una solución factible para dar solución a algunas de las limitaciones que actualmente posee el sistema.

Esta etapa es conocida como validación y consiste evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados. (43)

3.2.1 Resultados de las pruebas realizadas por la suite

Para comprobar que la suite desarrollada cumple con las especificaciones del usuario se realizó el proceso de validación, teniendo en cuenta las exigencias iniciales del sistema y diferentes valores de entradas que estuvieran en correspondencia con los validadores del módulo en cuestión. Se obtuvieron los siguientes resultados:

Módulo Administración	
Cantidad de casos prueba	68

Cantidad de pruebas	373
Satisfactorias	371
Insatisfactorias	1- Prueba realizada al campo “name” de la clase “Usuarios”. 2- Prueba realizada al campo “lastname” de la clase “Usuarios”.

Tabla 3. Resumen de las pruebas realizadas al módulo Administración del COJ.

En el caso de las pruebas realizadas a los campos “name” y “lastname” de la clase “Usuarios”, los errores detectados se deben a un mal comportamiento del validador, ya que este especifica que no se admiten caracteres numéricos y sin embargo permite añadir nombres o apellidos de esta forma. Para un mayor entendimiento a continuación se muestran dichas pruebas. (Las restantes se encuentran en el Anexo # 3)

3.2.2 Resultados de las pruebas para la clase Usuarios_Test

Campo: Name

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
A	si	si
Asdasdasdasdasdasdasdasdasdasd	si	si
María Fernanda	si	si
.-' .-'	si	si
654654654	no	si
#$%^&*()%^&*	no	no
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Name” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Insatisfactorio		

Tabla 4. Prueba realizada al campo Name de la clase Usuarios_Test.

Campo: Lastname

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
A	si	si
asdasdasdasdasdasdasdasdasdasdsdasdasdasdasdasdasdasdas	si	si
María Fernanda	si	si
.-' .-'	si	si
654654654	no	si
Maria/Fernanda	no	si
#$%^&*()%^&*	no	no
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Lastname” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Insatisfactorio		

Tabla 5. Prueba realizada al campo Lastname de la clase Usuarios_Test.

3.2.3 Resultados de las pruebas para la clase AdicionarConcurso_Test

Campo: Id

AdicionarConcurso_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
12332141	si	si
“”	no	no
“ “	no	no
As	no	no
12as	no	no
/*-	no	no
Descripción: Se desea comprobar si el validador del campo “Id” de la clase “ConcursosReales” en el paquete “Competencias” funciona correctamente.		

Evaluación de la prueba: Satisfactorio

Tabla 6. Prueba realizada al campo Id de la clase AdicionarConcurso_Test.

3.2.4 Resultados de las pruebas para la clase CodigoSitio_Test

Campo: Codigo

CodigoSitio_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
123	si	si
123abc	si	si
abc	si	si
89-a	si	si
ab-2	si	si
“”	no	no
89*	no	no
abc*	no	no
/*	no	no
Descripción: Se desea comprobar si el validador del campo “Codigo” de la clase “Sitio” en el paquete “COJBoard” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 7. Prueba realizada al campo Codigo de la clase CodigoSitio_Test.

3.3 Comportamiento de las métricas de mantenimiento, valoración mediante la técnica de Iadov

Para valorar la satisfacción respecto a la utilidad y aplicabilidad del sistema en entornos reales se empleó la técnica Iadov, que permite el estudio del grado de satisfacción del personal involucrado en un proceso objeto de análisis.

Primeramente se aplicó una encuesta de satisfacción a varias personas (ver Anexo # 4), teniendo en cuenta su familiarización con el sistema y con la propuesta de solución, además de su participación en el proceso de pruebas que se lleva a cabo como parte de la etapa de mantenimiento del COJ.

En este caso, la técnica Iadov está compuesta por cinco preguntas: tres cerradas y dos abiertas, las cuales se reformulan en la investigación para valorar el grado de satisfacción sobre la suite de pruebas. Para su elaboración se tuvieron en cuenta algunos indicadores de mantenimiento (definidos en el epígrafe 2.5) con que debe cumplir la solución propuesta. Una vez establecidas las preguntas se conforma el "cuadro lógico de Iadov":

El número resultante de la interrelación de las tres preguntas, indica el nivel de satisfacción de los sujetos, que se expresa en la escala numérica que oscila entre +1 y - 1 tal y como se muestra en la Tabla 8. Posteriormente se calcula el índice de satisfacción grupal (ISG) según la Ecuación 1, utilizando los diferentes niveles de satisfacción.

Nivel de satisfacción	Cantidad
Clara satisfacción	+1
Más satisfecho que insatisfecho	+0,5
No definido y contradictorio	0
Más insatisfecho que satisfecho	-0,5
Clara satisfacción	-1

Tabla 8. Niveles de satisfacción expresados en la escala numérica.

La satisfacción grupal se calcula por la siguiente fórmula:

$$ISG = \frac{A(+1) + B(+0{,}5) + C(0) + D(-0{,}5) + E(-1)}{N}$$

Ecuación 1. Cálculo del Índice General de Satisfacción.

Luego este valor es representado en el eje numérico siguiente:

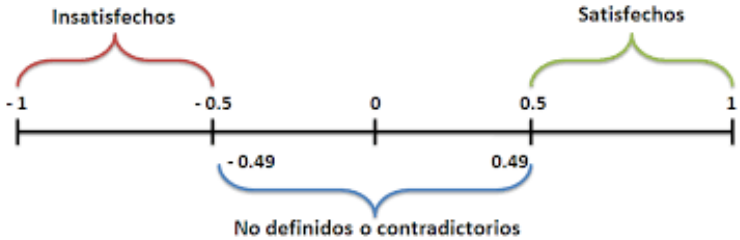

Ilustración 19. Rangos de valoración del ISG.

Los valores que se encuentran comprendidos entre - 1 y - 0,5 indican insatisfacción; los comprendidos entre - 0,49 y + 0,49 evidencian contradicción y los que caen entre 0,5 y 1 indican que existe satisfacción.

3.3.1 Resultados de la técnica de Iadov

¿Te gusta que la solución al problema planteado sea una suite de pruebas automatizadas?	¿La utilización de la suite de pruebas automatizadas para el COJ cumple con todas la funcionalidades requeridas?								
	NO			NO SÉ			SI		
	¿Consideras que la suite de pruebas contribuye al mantenimiento del COJ?								
	SI	NO SÉ	NO	SI	NO SÉ	NO	SI	NO SÉ	NO
Me gusta mucho	1	2	6	2	2	6	6	6	6
No me gusta mucho	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé qué decir	2	3	6	3	3	3	6	3	4

Ilustración 20. Cuadro lógico de Iadov elaborado.

Los resultados de la satisfacción individual según las categorías empleadas fueron los siguientes:

Nivel de satisfacción	Cantidad	%
Clara satisfacción	6	85,71
Más satisfecho que insatisfecho	1	14,29
No definido y contradictorio	0	0,00
Más insatisfecho que satisfecho	0	0,00
Clara satisfacción	0	0,00

Tabla 9. Resultado de aplicación de la técnica Iadov.

Sustituyendo en la ecuación del ISG

$$ISG = \frac{6(+1) + 1(+0{,}5) + 0(0) + 0(-0{,}5) + 0(-1)}{7}$$

$$ISG = \frac{6{,}5}{7}$$

$$ISG = 0{,}93$$

Ubicación del ISG en el eje numérico

Ilustración 21. Ubicación del ISG en el eje numérico.

Conclusión

Al procesar las respuestas a las encuestas en el cuadro lógico de Iadov, se obtiene un grado de satisfacción grupal de 0,93; lo cual se traduce en una clara satisfacción con el uso de la suite de pruebas automatizadas en el COJ para contribuir a su mantenimiento. En el criterio de los encuestados respecto a que la suite cumple con todas las funcionalidades requeridas, hubo una concordancia de un 100,00%. Respecto a la posibilidad de contribuir al mantenimiento del sistema existió una concordancia de un 100%. En cuanto a que es acertada la solución planteada, el 57,14% consideró que les gusta la propuesta, el 28,57% respondió que no le agrada del todo y el 14,29% lo calificó como *"Me da lo mismo"*.

Las preguntas abiertas que se formularon fueron:

- ✓ ¿Qué posibilidades de aplicación tiene la solución?
- ✓ ¿Utilizaría usted la solución propuesta para contribuir al proceso de automatización de pruebas en otros proyectos?

3.4 Características de la suite de pruebas automatizadas

De manera general, la suite de pruebas automatizadas para el COJ cumple con los siguientes criterios:

Criterios	Resultados del desarrollo de la suite
Corrección	Cumple con las funciones requeridas y se realizaron de forma correcta.
Usabilidad	Es "amigable al usuario", debido a que resulta fácil la interacción y comprensión del sistema.
Flexibilidad	Permite realizar modificaciones en el código con mayor facilidad.
Exactitud	Muestra exactamente el nivel de corrección de los test, expresándolo en porciento.
Concisión	Con el uso de JUnit y Mockito fue posible reducir la cantidad de líneas de código.
Consistencia	Se diseñaron uniformemente todas las pruebas teniendo en cuenta varios criterios y se realizó la documentación del código.
Rapidez	Rápida localización de errores y su modificación.

Tabla 10. Características de la suite de pruebas automatizadas para el COJ.

3.5 Resultados finales de la investigación

Antes del desarrollo de la suite, el sistema no contaba con un mecanismo para comprobar el correcto funcionamiento de sus módulos de códigos, resultando más propensa a la aparición de errores, lo que representaba una limitación en el correcto funcionamiento y rapidez de integración que requiere. Además, no era posible determinar si el sitio cumplía con los requerimientos del cliente. También resultaba difícil la localización y la corrección de los errores. Todas estas deficiencias afectaban directamente la posibilidad de elevar el mantenimiento del sistema.

Luego de la implementación de la suite y analizados los resultados obtenidos con la aplicación de la técnica de Iadov para medir la satisfacción con respecto a la misma, se puede afirmar que se logró contribuir al mantenimiento del Juez en Línea Caribeño, favoreciendo grandemente el proceso de pruebas de software en dicho sistema.

3.6 Conclusiones del capítulo

Con la realización de este capítulo se logró desarrollar la correcta implementación de la suite mediante la utilización del estándar de codificación *CamelCase* y con el uso del code coverage fue posible determinar la cantidad de código que está siendo cubierto por las pruebas. La aplicación de la técnica de Iadov permitió medir el nivel de satisfacción con respecto solución, teniendo en cuenta el criterio de varias personas relacionadas con el sistema y proceso de pruebas que se lleva a cabo en el COJ.

Conclusiones generales

Después de realizada la investigación, identificados los objetivos y culminado el cumplimiento de las tareas y la solución al problema planteado, se puede concluir lo siguiente:

- ✓ Luego de establecer los referentes teóricos y metodológicos de los temas asociados a las pruebas automatizadas y el mantenimiento de software, fue posible la selección de las herramientas JUnit, Mockito, IntelliJ IDEA y Selenium IDE que facilitaron la creación de la suite de pruebas automatizadas para el COJ.
- ✓ A partir de las limitaciones del actual proceso de pruebas del COJ y estructurado el diagrama de paquetes correspondiente al módulo Administración, se logró desarrollar la suite de pruebas automatizadas para dicho sistema.
- ✓ Con el uso de la técnica de Iadov fue posible medir el nivel de satisfacción existente con respecto a la solución, teniendo en cuenta el impacto de la misma en el proceso de pruebas que se realiza como parte de la etapa de mantenimiento del COJ, enfocado en la completitud de la suite y su importancia para el sistema.

Recomendaciones

Para dar continuidad a la presente investigación, se recomienda:

- ✓ Extender el uso de las pruebas automatizadas en los restantes módulos del COJ que contribuyan a mejorar su etapa de mantenimiento.

Referencias

1. **Pressman, Roger S.** *Ingeniería de software. Quinta edición.* 2007.

2. **Informáticos, Departamento de Lenguajes y Sistemas.** *BLOQUE II: Integración de Sistemas Software. .* Sevilla : s.n., 2013.

3. **PBWORK. [En línea] http://isg2.pbworks.com/w/page/7624280/Pruebas%20del%20Software.**

4. Esteve Ambrosio, Daniel Alberto. ***Implantación de un proceso de automatización de pruebas para una aplicación software.*** **Valencia : s.n., 2015.**

5. Menéndez Gómez, Yurisel . ***Pruebas Automatizadas para el plug-ins Adquisición de Datos de la Interfaz Hombre – Máquina.*** **2011. Tesis.**

6. ***SG Buzz. Beneficios de la Automatizacíon de Pruebas.*** **[En línea] 8 de junio de 2016. [Citado el: 8 de junio de 2016.] http://sg.com.mx/content/view/683.**

7. Oterino, Ana M. del Carmen García. javiergarzas.com. [En línea] 2015. http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html.

8. Garzas, Javier. Diferencias entre los diferentes tipo de pruebas. [En línea] 4 de julio de 2014. http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html.

9. Curso .NET con C#. Pruebas unitarias. [En línea] enero de 2015. http://si.ua.es/es/documentacion/c-sharp/documentos/pruebas/07pruebasunitarias.pdf.

10. Ingeniería de Software. ***Universidad Nacional Abierta y a Distancia.*** **[En línea] 2014. http://datateca.unad.edu.co/contenidos/301404/301404_ContenidoEnLinea/leccin_43__prueba_de_integracin.html.**

11. Técnicas de Evaluación Dinámica. [En línea] 2014. http://www.lsi.us.es/docencia/get.php?id=361.

12. Pérez Lamancha, Beatriz. ***Gestión de las Pruebas Funcionales.*** **Montevideo, Uruguay : Centro de Ensayos de Software.**

13. León, Marcela. KIBERNUM. [En línea] 15 de septiembre de 2015. http://www.kibernum.com/noticias/por-que-son-importantes-las-pruebas-funcionales-2/.

14. Briceño, Gary. Automatización de pruebas de aceptación. [En línea] [Citado el: 3 de junio de 2016.] http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwiWn5HmsJzNAhXIXR4KHU26AJUQFgggMAE&url=http%3A%2F%2Fwww.gary.pe%2Fblog%2F2015%2Fautomatizacion-de-pruebas-de-aceptacion%2F&usg=AFQjCNEmQlLukJLkD9TqDPH1loL52sltMw&sig2=4rmEAwLzXvqk.

15. Letelier, Patricio. ***Pruebas de Aceptación comoconductor del Proceso Software.*** **Valencia : s.n.**

16. Engineers, The Institute of Electrical and Electronics. IEEE Standard for Software. New York : s.n., 1998.

17. Moral García, Santiago. *Mantenimiento del Software.* s.l. : Grupo Kybele, 2012.

18. Novella, Mª Ángeles López, y otros. *Marco de evaluacion EFQN.* 2006.

19. Pressman, Roger S. *Mantenimiento de software.* 1998.

20. Francisco Ruiz, Macario Polo. *Mantenimiento del Software.* Ciudad Real : Grupo Alarcos, 2001.

21. *Ingeniería de Software. Manteniemiento de Software.* 2010.

22. López Quesada, Juan Antonio. *Mantenimiento del Software.*

23. Francisco Ruiz, Macario Polo. *Mantenimiento del Software.* Ciudad Real : s.n.

24. Lenguajes de programación. *CMM.* [En línea] julio de 2014. http://es.ccm.net/contents/304-lenguajes-de-programacion.

25. Lenguajes de programación. Programación Java. [En línea] 2016. [Citado el: 25 de febrero de 2016.] http://www.lenguajes-de-programacion.com/programacion-java.shtml.

26. ¿Qué es un 'framework'? . [En línea] 2015. http://jordisan.net/blog/2006/que-es-un-framework/.

27. Blé Jurado, Carlos. *Diseño Ágil con TDD.* 2010.

28. Jiménez Centeno, Germán. Adictos al trabajo. [En línea] http://www.adictosaltrabajo.com/tutoriales/mockito/.

29. Herramientas test software. [En línea] 28 de febrero de 2010. http://www.nosolounix.com/2010/02/herramientas-test-software.html.

30. Marco de Desarrollo de la Junta de Andalucía. [En línea] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/381.

31. Crimson solutions. [En línea] 2015. http://www.crimsonsolutions.co.uk/testing/testing-downloads.php.

32. Herramientas para pruebas de software. [En línea] 2 de enero de 2015. http://testeandosoftware.com/10-herramientas-para-pruebas-de-software-ii/.

33. WebInject-Open Source Web/ HTTP Test Tool. [En línea] 2006. [Citado el: 8 de junio de 2016.] http://www.webinject.org/.

34. Entorno de Desarrollo Integrado (IDE). [En línea] 25 de enero de 2013. https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

35. IntelliJ IDEA. Capable and Ergonomic Java * IDE. [En línea] 2016. https://www.jetbrains.com/idea/.

36. EcuRed. *Automatización de pruebas.* [En línea] junio de 2016. http://www.ecured.cu/Automatizaci%C3%B3n_de_pruebas.

37. Ingeniería de software. [En línea] 8 de julio de 2015. http://jraquelm2.wix.com/ingenieriadesoftware#!-TEMA-9-DIAGRAMAS-DE-PAQUETES-Y-DE-SECUENCIAS/cmbz/55a540f80cf25b8bf7e9279f.

38. Mockito. [En línea] 28 de diciembre de 2009.

39. Ingeniería de Software III. [En línea] 24 de noviembre de 2012. http://ing-software3.blogspot.com/2013/01/metricas-del-mantenimiento.html.

40. Rodríguez, Tom. SileShare. *Desarrollo e implantación del software de aplicación.* [En línea] 20 de septiembre de 2011. http://es.slideshare.net/TomRodriguez/implementacion-de-software.

41. Paez, Nico. Code Coverage. [En línea] 2016. https://blog.nicopaez.com/2012/01/14/code-coverage/.

42. Armesto, Jose . El problema con el code coverage. [En línea] 2015. http://blog.armesto.net/el-problema-con-el-code-coverage/.

43. Gutiérrez, Javier. *Introducción al Proceso de Pruebas.* 2011.

Anexos

Anexo # 1. Encuesta sobre pruebas automatizadas

Encuesta para obtener información sobre el conocimiento de las pruebas de software y las herramientas automatizadas para tal fin.

Elaborada por: Wendys de la Caridad Jiménez Estévez

Franly Hernández Chávez

Con el objetivo de evaluar el nivel de conocimiento que existe en cuanto al proceso de pruebas de software, las herramientas automatizadas que se utilizan en ellas y por la importancia que tiene dicho proceso en el desarrollo de software; necesitamos su colaboración a través de esta encuesta para recoger su opinión y conocimiento acerca del tema.

De antemano le agradecemos su colaboración y el tiempo que ha dedicado a la misma. Gracias.

1. ¿Conoce usted qué es una prueba automatizada de software?

   _____ SI _____ NO

   En caso afirmativo diga:

   a. Cuáles de estas pruebas que se exponen a continuación conoce. Marque con una (X).

      _____ Pruebas unitarias

      _____ Pruebas funcionales

      _____ Pruebas de regresión

      _____ Pruebas de aceptación

      _____ Pruebas de integración

2. ¿Se realizan en su centro algunas de ellas?

   _____ SI _____ NO

   a. ¿Cuáles considera que se realizan en su centro?

_____ Pruebas unitarias

_____ Pruebas funcionales

_____ Pruebas de regresión

_____ Pruebas de aceptación

_____ Pruebas de integración

3. ¿Conoce qué es una herramienta de prueba?

_____ SI _____ NO

4. ¿Se utiliza en su centro alguna herramienta de pruebas funcionales?

_____ SI _____ NO

5. ¿Conoce alguna de las herramientas de pruebas funcionales que se enuncian a continuación? Marque con una X.

_____ Selenium

_____ Solex

_____ WebCorder

_____ Wireshark

_____ Imprimatur

_____ WET

_____ Capedit

_____ WebInject

6. Usted considera que la realización de pruebas funcionales permite:

_____ Detectar errores con mayor rapidez.

_____ Obtener un software con mayor calidad.

_____ Validar que el software hace lo que se ha especificado.

Anexo # 2. Resultados de la encuesta sobre pruebas automatizadas

En la figura 16 de la pregunta 2 se evidencia un 100 % de utilización de las pruebas de software en cada área donde se realizó la encuesta.

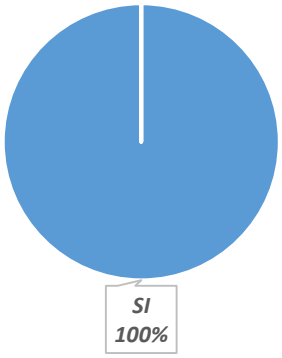

Ilustración 22. . Aplicación de pruebas automatizadas en el centro.

En la figura 17 de la pregunta 3 se observa cómo el 100% conoce qué son las herramientas de prueba.

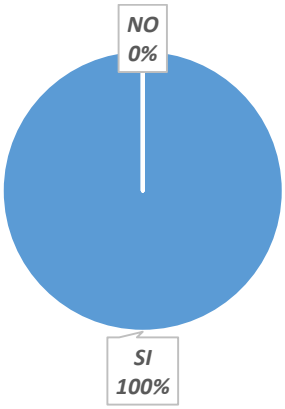

Ilustración 23. Conocimiento de herramientas de prueba.

En la figura 18 que corresponde a la pregunta 4, se evidencia el uso o no de las herramientas de pruebas funcionales en las diferentes áreas, obteniendo como resultado que solo un 6 % de los encuestados no las utiliza.

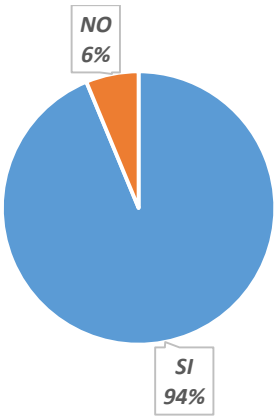

Ilustración 24. Utilización de herramientas para pruebas funcionales.

Anexo # 3. Resultados de las pruebas para cada clase

Clase Usuarios_Test

Campo: Nick

Usuarios_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
asd	si	si
12341234	si	si
[];'./@#$%^	si	si
asd	si	si
Asd	si	si
asd asd 333	si	si
123123123123123	si	si
“”	no	no
“ “	no	no
a	no	no
123456789123456sasdasdGGGs	no	no
Descripción: Se desea comprobar si el validador del campo “Nick” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 11. Prueba realizada al campo Nick de la clase Usuarios_Test.

Campo: Country_id

Usuarios_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
14	si	si
-1	si	si
0	no	no
Descripción: Se desea comprobar si el validador del campo “Country_id” de la clase “Usuarios” funciona correctamente.		

Evaluación de la prueba: Satisfactorio

Tabla 12. Prueba realizada al campo Country_id de la clase Usuarios_Test.

Campo: Institution_id

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
564654888	si	si
1	si	si
-4545	si	si
-1	no	no
Descripción: Se desea comprobar si el validador del campo “Institution_id” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 13. Prueba realizada al campo Institution_id de la clase Usuarios_Test.

Campo: Locale

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
1	si	si
2	si	si
-5	si	si
687987465	si	si
Descripción: Se desea comprobar si el validador del campo “Locale” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 14. Prueba realizada al campo Locale de la clase Usuarios_Test.

Campo: Lid

Usuarios_Test

Casos de Pruebas	Respuesta Esperada	Respuesta Real
1	si	si
2	si	si
-5	si	si
0	si	si
687987465	si	si
0	no	no
Descripción: Se desea comprobar si el validador del campo “Lid” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 15. Prueba realizada al campo Lid de la clase Usuarios_Test.

Campo: Password

Usuarios_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
798654 4565 asdasd asdasd	si	si
798654 4565	si	si
!@#$%^&*()_	si	si
MAMAMAMasas121 +_)*&^%$#	si	si
qwertyui	si	si
qwe45lkjhgasrtyuiopasdfghjklzxcvbnm,qwerftyuiopasdfghjklzxcvbnmqwertyuioasdfghjkxcvbnsdfghjkldvbnm	si	si
a	no	no
asdasdasdasdasdasdasdasdasdasdasdasdasdasdasdasdaksjdhakjsbdkjahsdkjahsdkjhakjsdhakjsdhkjasdhakjsdhkajsdh	no	no
798654	no	no
“”	no	no
“ “	no	no

Descripción: Se desea comprobar si el validador del campo “Password” de la clase “Usuarios” funciona correctamente.
Evaluación de la prueba: Satisfactorio

Tabla 16. Prueba realizada al campo Password de la clase Usuarios_Test.

Campo: Access_rule

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
*	si	si
0	no	no
,	no	no
qwe	no	no
-54654	no	no
&*	no	no
*65	no	no
*	no	no
*	no	no
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Access_rule” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 17. Prueba realizada al campo Access_rule de la clase Usuarios_Test.

Campo: Gender

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
1	si	si
2	si	si

Descripción: Se desea comprobar si el validador del campo “gender” de la clase “Usuarios” funciona correctamente.
Evaluación de la prueba: Satisfactorio

Tabla 18. Prueba realizada al campo Gender de la clase Usuarios_Test.

Campo: Email

Usuarios_Test		
Casos de Pruebas	***Respuesta Esperada***	***Respuesta Real***
asd@asd.asd	si	si
fchavez@uci.cu	si	si
asd132@asd.asd	si	si
asd_132@asd.asd	si	si
fchavez@uci.cu%	si	si
fchavez@uci	no	no
asd@asd.asd.asd.asd.asd.asd.	no	no
asd 123@asd.asd	no	no
“”	no	no
“ ”	no	no
fchavez@@uci	no	no
fchavez @uci.cu	no	no
fchavez@ uci.cu	no	no
fchavez @uci. cu	no	no
chavez@uci.cu	no	no
fchavez @uci.cu	no	no
fchavez@uci.cu%*a%s^d	no	no
Descripción: Se desea comprobar si el validador del campo “Email” de la clase “Usuarios” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 19. Prueba realizada al campo Email de la clase Usuarios_Test.

Clase NombreEtiqueta_Test

Campo: Nombre

NombreEtiqueta_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
Nombre	si	si
Nombre	si	si
Nombre123	si	si
Nombre1234 -*/	si	si
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Nombre” de la clase “Etiquetas” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 20. Prueba realizada al campo Nombre de la clase NombreEtiqueta_Test.

Clase NombreFuente_Test

Campo: Fuente

NombreFuente_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
Fuente	si	si
Fuente	si	si
Fuente	si	si
Fuente Fuente	si	si
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Fuente” de la clase “Fuentes” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 21. Prueba realizada al campo Fuente de la clase NombreFuente_Test.

Clase AutorFuente_Test

Campo: Autor

AutorFuente_Test		
Casos de Pruebas	**Respuesta Esperada**	**Respuesta Real**
Autor	si	si
Autor	si	si
Autor	si	si
Autor Autor	si	si
“”	no	no
“ “	no	no
Descripción: Se desea comprobar si el validador del campo “Autor” de la clase “Fuentes” funciona correctamente.		
Evaluación de la prueba: Satisfactorio		

Tabla 22. Prueba realizada al campo Autor de la clase AutorFuente_Test.

Anexo # 4. Encuesta para medir el nivel de satisfacción

Elaborada por: Wendys de la Caridad Jiménez Estévez

Franly Hernández Chávez

Con el objetivo de evaluar el nivel de importancia que se le concede a la suite de pruebas automatizadas desarrolladas para el Juez en Línea Caribeño, se decide aplicar la siguiente encuesta de satisfacción a varios profesores, teniendo en cuenta su familiarización con el sistema y con la propuesta de solución, además de su participación en el proceso de pruebas que se lleva a cabo como parte de la etapa de mantenimiento de dicho sistema.

De antemano le agradecemos su colaboración y el tiempo que ha dedicado a la misma. Gracias.

Según su criterio, responda las siguientes preguntas:

1. ¿La utilización de la suite de pruebas automatizadas para el COJ cumple con todas la funcionalidades requeridas?

___Sí ___No ___No sé

2. ¿Consideras que la suite de pruebas contribuye al mantenimiento del COJ?

___Sí ___No ___No sé

3. ¿Te gusta que la solución al problema planteado sea una suite de pruebas automatizadas? Especifique el grado de satisfacción.

____Me gusta mucho

____No me gusta mucho

____Me da lo mismo

____Me disgusta más de lo que me gusta

____No me gusta nada

____No sé qué decir

4. ¿Qué posibilidades de aplicación tiene la solución?

____Muchas

____Pocas

____Ninguna

5. ¿Utilizaría usted la solución propuesta para contribuir al proceso de automatización de pruebas en otros proyectos?

___Sí ___No ___No sé