UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E
INGENIERÍA DE LA COMPUTACIÓN

DIRECTRICES PARA EL DESARROLLO DE SOFTWARE CIENTÍFICO

TESIS
QUE PARA OPTAR POR EL GRADO DE
Maestra en Ciencias e Ingeniería en Computación

PRESENTA:
Ariadne Olarte Cetina

TUTOR:
María Guadalupe Elena Ibargüengoitia Gonzáles
Facultad de Ciencias

Ciudad Universitaria, CDMX, Julio de 2021

Tabla de contenido

INTRODUCCIÓN **3**

Objetivo del trabajo 4
Hipótesis de trabajo 4
Contribución y Relevancia 4
Metodología de trabajo 5

CAPÍTULO 1. SOFTWARE CIENTÍFICO **6**

Antecedentes 6
El Software Científico en las investigaciones 7
Definición Software Científico 8
Características del Software Científico 8
Criterios de Calidad: 10
Pruebas en el Software Científico 10
Validación de Software Científico 11
Documentación en el Software Científico 12

CAPÍTULO 2 ESTÁNDARES Y HERRAMIENTAS EN INGENIERÍA DE SOFTWARE **13**

CCMI para desarrollo 13
ISO 15504 de calidad del software 13
ISO 12207 procesos de ciclo de vida de software 13
ISO 29110 procesos de software en las pequeñas organizaciones (..... 13
Esencia 14

CAPÍTULO 3. EXPLORACIÓN DEL USO DE LA INGENIERÍA DE SOFTWARE EN EL DESARROLLO DE SOFTWARE CIENTÍFICO **16**

CAPÍTULO 4: PROPUESTAS DE DIRECTRICES PARA EL DESARROLLO DE SOFTWARE CIENTÍFICO. ÁREA DE CLIENTE **21**

Conceptos teóricos 21
Oportunidad 22
Involucrados 23
Investigadores 25
Programadores 25
Testers 26
Administradores 26

CAPÍTULO 5 ÁREA DE SOLUCIÓN **28**

5.1 Requerimientos 28
5.1 Herramientas para la obtención de Requerimientos 28
Validación *29*
Priorización *30*
5.3 Directrices para la definición de los requerimientos 32

CAPÍTULO 6. SISTEMA DE SOFTWARE **34**

6.1 Diseño o modelado del sistema de alto nivel 34
6.1.1 Diagramas de paquetes y su modelado UML *35*
6.1.2 Directrices para el modelado del sistema *40*
Directrices para el diagrama de paquetes *40*

6.2 CONSTRUCCIÓN 40
6.2.1 Definir ambiente de construcción *40*
6.2.2 Conceptos de Frameworks *41*
Spring Framework *41*
Django Framework *42*
6.2.3 Manejo de Versiones *42*
6.2.4 CONFIGURACIÓN DEL AMBIENTE DE IMPLEMENTACIÓN ELEGIDO 43
SPRING 43
DJANGO 45
6.2.5 MANEJADOR DE VERSIONES GITHUB 49
6.2.6 MANEJADOR DE BASES DE DATOS 52
6.3 DEFINIR AMBIENTE DE CONSTRUCCIÓN 59
6.3.1 Directrices para definir el ambiente de desarrollo *59*
6.3.2 Directrices para el control de versiones *59*
6.4 PRUEBAS 60
Pruebas unitarias *61*
Pruebas de caja blanca *61*
Pruebas de Integración *62*
PRUEBAS DE SISTEMA 62
Pruebas de regresión *62*
6.4.1 Herramienta para la gestión de prueba *63*
6.4.2 Directrices para pruebas *66*
Definición de pruebas unitarias *66*
Definición de pruebas de integración *68*
Definición de pruebas de regresión *70*
Gestión de pruebas *72*
6.5 MANTENIMIENTO Y EVOLUCIÓN 72
6.5.1 Herramientas para el mantenimiento o evolución *72*
6.5.2 Directrices para manejar la evolución del software *73*

CAPÍTULO 7. VALIDACIÓN DE PROPUESTA POR EXPERTOS Y RESULTADOS OBTENIDOS. 74

RESPUESTAS AL CUESTIONARIO PARA LA VALIDACIÓN 74

CONCLUSIONES Y TRABAJO FUTURO. 83

BIBLIOGRAFÍA 84

ANEXOS 85

Introducción

El software ha permitido a sus usuarios automatizar procesos, realizar cálculos, modelar problemas, busca facilitar cualquier tarea a través de la ejecución de un conjunto de procesos computacionales, incluso ayudó al hombre a llegar a la luna en 1969 gracias al desarrollo de software de navegación “on-board” para el programa espacial Apolo (MIT, 2016).

En 2020 las personas tienen en sus bolsillos software incluso más complejo que el que llevó al hombre a la luna y sin embargo su propósito es similar, ayudar a facilitar tareas que para los usuarios resultan tediosas.

La forma en que se desarrolla software también ha evolucionado a lo largo de los años iniciando por el desarrollo de prueba y error hasta que surgieron metodologías y estándares de desarrollo de software que dan una visión más concreta de cómo desarrollar software. Pero no es lo mismo desarrollar software para una calculadora que un software que se utilizará en un cohete, la diferencia es abismal conociendo simplemente el objetivo de este software.

El Software Científico puede considerar los estándares y metodologías creados para el desarrollo de software comercial adaptándolos a sus propias necesidades. Al igual que el software en general, la forma en la que se hace ciencia está en constante cambio y el software constituye parte importante de las investigaciones científicas.

Tener un conjunto de directrices que se enfoquen en Software Científico puede ayudar a los investigadores, cuya formación no es necesariamente la misma que un ingeniero de software, a tener un camino establecido en el desarrollo de su propio software que podría ayudar a obtener un producto de software de calidad, que apoye en el resultado final de toda la investigación.

El software de calidad puede facilitar la vida de sus usuarios, pero para lograr un software de calidad y eficiente es necesario que su desarrollo sea constantemente supervisado y basado en un método establecido basado en buenas prácticas para guiar a los programadores durante el proceso de desarrollo.

En el caso de Software Científico ocurre lo mismo, tener una metodología de desarrollo es vital pues dicho software ayudará a probar la hipótesis de la investigación, así como en la automatización, redacción, ejecución de pruebas, simulación de posibles soluciones, análisis de resultados etc.

Objetivo del trabajo

Desarrollar directrices enfocadas al ciclo de vida en el desarrollo de Software Científico, que permita a los desarrolladores saber cómo se debe construir el software de calidad, que resuelva las necesidades de los interesados y que logre responder a las preguntas de investigación planteadas. En el caso del Software Científico es sumamente importante tener un ciclo de desarrollo basado en estándares internacionales para el desarrollo de software, de la misma forma que cualquier investigación científica.

Hipótesis de trabajo

La hipótesis de este trabajo es:

¿El uso de directrices de Ingeniería de Software concebidas para el desarrollo de Software Científico ayuda a los desarrolladores a generar documentación y software de calidad?

Dentro del proceso de desarrollo de Software Científico no es prioritario el uso de prácticas de Ingeniería de Software, por lo que aspectos como la planeación, el diseño o la documentación pasan a un segundo plano, sin embargo, es bien sabido que utilizar estándares para el desarrollo mejora la calidad del software construido y el proceso de construcción se vuelve más claro. Por lo que generar directrices que permitan mejorar el proceso de desarrollo y que permitan generar documentación del Software Científico es importante, pues este tipo de software debe ser tratado con la misma seriedad con la que se lleva una investigación científica.

Contribución y Relevancia

Los investigadores que crean software para apoyar sus investigaciones no siempre poseen el conocimiento sobre Ingeniería de Software y los estándares que la rigen, por lo que muchas veces avanzan en sus proyectos conforme los requerimientos aparecen.

Sin embargo, es necesario tratar el desarrollo de Software Científico de manera tan formal y cuidadosa como cualquier investigación científica para la cual sea desarrollado, ya que errores del software podrían provocar resultados no esperados en la investigación, incluso ponerla en duda.

Por lo que en este trabajo se busca desarrollar conjunto de directrices para el ciclo de vida del Software Científico que permitirá conocer mejor la forma en cómo generar este tipo de software de manera formal. Definir lineamientos y normas de cómo debe ser tratado el software para la investigación, permitirá a los investigadores hacer software de calidad que permita respaldar los resultados obtenidos

Metodología de trabajo

1. Establecer las características peculiares del Software Científico
2. Revisar la existencia de estándares y herramientas a utilizar en el desarrollo de proyectos de Software Científico
3. Investigar sobre el tipo de pruebas a realizar en el Software Científico.
4. Investigar/Conocer los requerimientos de la documentación de una investigación científica apoyada por Software Científico.
5. Proponer directrices para el desarrollo de Software Científico.
6. Proponer directrices para la documentación de Software Científico.
7. Validar las propuestas, verificando que tan efectivas han sido al aplicarlas en un proyecto real.

La estructura de este trabajo es la siguiente:

- Software Científico: en este capítulo se identifican las características más importantes del Software Científico, criterios de calidad y la importancia de una documentación adecuada, incluye antecedentes de la forma en cómo se lleva el desarrollo de este tipo de software.

- Estándares y Herramientas en Ingeniería de Software: en este apartado se muestran algunos de los estándares más importantes en el desarrollo de software cuyos fundamentos permiten el desarrollo de la propuesta en este trabajo, validación y pruebas dentro del Software Científico.

- Exploración de uso de la Ingeniería de Software en el desarrollo de Software Científico: contiene información relacionada con el uso actual de la Ingeniería de Software para el desarrollo de Software Científico, identificar como se han aplicado dichos estándares actualmente.

- Propuestas de directrices para el desarrollo de Software Científico: contiene las propuestas de directrices que pueden ser utilizadas para mejorar el proceso de desarrollo en el Software Científico, definición de los roles y actividades que se realizan.

Capítulo 1. Software Científico

Antecedentes

Las metodologías de desarrollo han sido creadas en su mayoría para software más comercial dentro de la industria tal es el caso del ISO-29110 para pequeñas organizaciones o el modelo CMMI que mejora de procesos de una organización completa. Al tratarse de Software Científico estas metodologías no contemplan todas las características que involucra este tipo de software, cuando se desarrolla Software Científico no solamente se espera que el software resuelva una necesidad del cliente, sino que se debe probar que el proceso que se sigue el código para dar una solución es el adecuado, así como generar documentación del proceso.

Es necesario reconocer que el Software Científico tiene ciertas particularidades como el buscar validar una hipótesis propuesta por los interesados, algoritmos especializados, y la necesidad por generar documentación que les permita respaldar esta serie de procesos y resultados obtenidos.

Durante la creación del Software Científico es poco común el uso de algunas prácticas de Ingeniera de Software que involucran estándares de calidad de software, en algunos casos se utilizan algunas herramientas que ayudan en la toma de decisiones tales como el uso de cuestionarios o votaciones (P Maller, 2012)., de igual forma se tiene nula evidencia de la existencia de estándares dedicados únicamente al desarrollo del Software Científico.

La Ingeniería de Software ha cobrado relevancia en el desarrollo de Software Científico (SC), puesto que algunos científicos han tenido que retractarse en sus publicaciones debido a fallas de software (U. Kanewala, 2014). El Software Científico presenta características y desafíos diferentes a los que se tienen cuando se desarrolla software comercial, una de las mayores diferencias se encuentra entre los desarrolladores científicos y los ingenieros de software.

Es importante que los desarrolladores aprendan a identificar las diferencias entre hacer Software Científico y software comercial, pues no tienen el mismo objetivo ni usuarios finales parecidos, incluso para crear Software Científico es necesario tener una base de conocimientos diferente que no solo está relacionada con la programación o los lenguajes para desarrollar software.

En los últimos años se ha buscado la mejora en Software Científico para que sea más rápido y barato de desarrollar, usando principalmente 2 estrategias.

1. Adoptando un marco de mejora de la industria como lo es CMMI (Mellon, 2018)

2. Enfoque botton-top “diseño de desarrollo enfocado en las pruebas tempranas, incluso en la porción de software más pequeña” (Llope M., 2017) evaluando problemas y prioridades en el equipo de desarrollo.

Se han desarrollado marcos de trabajo donde se identifican los puntos esenciales en la construcción de SC, como son la generación, introspección y priorización de los ítems o requerimientos que el software deberá cumplir para satisfacer la necesidad del desarrollo, utilizando técnicas como: Card Sorting (creación y clasificación en grupos de tarjetas) (McGeorge, 1997), Kelly Grids (Kelly, 1995) (introspección sobre una situación de cada miembro del equipo) y Multivoting (para una rápida priorización, donde cada participante podía votar entre las opciones de mejora propuestas en el SC) (Improvement, 2004), esta experiencia se realizó en equipos de desarrollo de software técnico-científico de la Dirección de Análisis Operativo de la Fuerza Aérea Argentina (P Maller, 2012).

Por medio de la implementación de procesos de Ingeniería de Software tales como documentación durante el ciclo de vida del desarrollo en el Software Científico, harán que el proceso sea sencillo de seguir para los investigadores a cargo de este desarrollo.

La documentación es importante en todo software, más aún si se trata de un software que es resultado de una investigación científica o que sirve como herramienta al momento de probar una hipótesis, este software puede servir como parte de la demostración de correctitud en los resultados de cualquier investigación.

El Software Científico en las investigaciones

El Software Científico (SC) ha cobrado una enorme importancia en las últimas décadas gracias al crecimiento del poder computacional.

Anteriormente era utilizado en su mayoría para agilizar los cálculos que eran muy largos para realizarse a mano por una sola persona donde la probabilidad de cometer errores era demasiado alta, se buscaba tener la mayor precisión en los cálculos, ya que de estos dependía si el valor de las acciones en la bolsa se mantenía o si el lanzamiento de un nuevo satélite de comunicaciones sería exitoso.

Organizaciones tan grandes como la NASA o Microsoft tenían en su poder los recursos y la tecnología necesaria para crear software especializado, algunas universidades invirtieron en equipo de cómputo que, en ese entonces era sumamente complicado manejar y cuyo costo era realmente alto, por lo que los investigadores y desarrolladores de ese tiempo tenían que solicitar permiso para utilizar estos equipos por un espacio de tiempo y realizar todas las pruebas que pudieran.

En 1958 la Universidad Nacional Autónoma de México puso en operación la primera computadora en México una IBM-6501 ubicada en el sótano de la Facultad de Ciencias de Ciudad Universitaria, entonces únicamente los académicos especializados podían obtener acceso a este equipo pues no era sencillo de utilizar. La IBM-6501 operaba con bulbos y era capaz de realizar la impresionante cantidad de 1,300 operaciones por segundo, funcionaba por medio de tarjetas perforadas por lo que se requerían conocimientos de cómo escribir

instrucciones en estas tarjetas que posteriormente la computadora sería capaz de interpretar y ejecutar (unam, 2008).

El poder de cómputo ha crecido exponencialmente mientras que en 1958 las computadoras realizaban un promedio de 1,300 operaciones por segundo, en la actualidad se pueden realizar un aproximado de 200 millones de operaciones por segundo, debido a este enorme crecimiento del cómputo y a la comercialización de estos equipos ahora es mucho más sencillo que cualquier investigador o académico tenga acceso a una computadora con los recursos suficientes que le permitan realizar cálculos sumamente complejos, analizar datos, modelar soluciones a problemas planteados etc.

Los investigadores han convertido al cómputo en parte importante de sus investigaciones, generando resultados de una manera más **rápida y precisa**, esto ha permitido que el computo se convirtiera en una parte fundamental para hacer ciencia, cada investigación científica tiene sus propios objetivos y características, sin embargo, construir una computadora especializada para cada una de las nuevas investigaciones seria excesivamente costoso y tardado, la solución fue el software.

Crear un software que se encargara de realizar tareas especificar y más tarde modificar para añadir procesos o eliminar los que ya no fueran necesarios, resultaría más sencillo que reinventar la computadora, de esta forma surgen nuevas aplicaciones especializadas en diversos campos de investigación, como astronomía, antropología, matemáticas, geología, física, biología, química, economía, etc.

Definición Software Científico

El Software Científico es la especificación en código de métodos y algoritmos (matemáticos, lógicos, estadísticos) por los cuales la información es categorizada o procesada para obtener resultados específicos, dichos métodos surgen de la hipótesis en una investigación. El Software Científico es desarrollado con la finalidad obtener resultados o generar modelos de comportamiento, al estar tan relacionado con el proceso de obtención de resultados debe pasar por un minucioso proceso de pruebas de tal forma que se pueda verificar que se ejecuta el proceso correcto. (Hinsen, 2013)

Considerado por algunos como el tercer pilar de la ciencia, después de la teoría y la experimentación, es visto como una herramienta que permite probar ideas durante una investigación, generar prototipos y reproducir resultados de una manera más rápida, es útil en la toma de decisiones, hacer predicciones, calcular y obtener evidencias, cuyo código es la especificación de cómo se resuelve el problema planteado, por lo que es considerado el equivalente a las fórmulas matemáticas.(Hinsen, 2013)

Características del Software Científico

Es importante conocer las diferentes características que existen entre el software que se desarrolla en la industria y el software creado con fines científicos, en la actualidad la mayor cantidad de software desarrollado es más comercial, desde aplicaciones bancarias hasta juegos.

Sin embargo, existen grandes diferencias entre el software comercial y el Software Científico, en este último es importante tener una importante base de conocimiento acerca de los procesos que se buscan modelar, así como de la hipótesis que se desea probar.

En el caso por ejemplo, de un software que simula el comportamiento de un virus es necesario conocer datos como la población que es más vulnerable ante este virus, tasas de contagio, cargas virales actuales, tiempo de vida del virus, vector de transmisión etc. Esta información es el resultado de una previa investigación científica y un software que modela la forma en cómo se transmitiría el virus bajo determinadas condiciones permite obtener información de cómo podría comportarse y en cuánto tiempo toda la población sería portadora del virus, esto permitiría tomar medidas para reducir la tasa de contagios y evitar una catástrofe.

Algunas características importantes que podemos encontrar en el Software Científico (P. Valenzuela-Toledo, 2018) son:

- **Desarrollo basado en Hipótesis**

Su desarrollo se basa en una hipótesis con **objetivos específicos**, los investigadores conocen el problema que se debe solucionar a través del software partiendo de una suposición o idea, conociendo el método por el cual debe resolverse, siendo estos métodos fórmulas matemáticas, procesos bioquímicos, análisis de datos etc.

A partir de esta base de conocimientos que poseen los investigadores construyen una idea que puede dar solución al problema planteado, posteriormente esta idea es transmitida, en algunos casos, a desarrolladores de software quienes se encargan de modelar dicha solución en software que es utilizado para comprobar sí la hipótesis planteada es correcta.

- **Alcance del software (cambios)**

Los resultados obtenidos pueden ser diferentes a lo esperado en la investigación, por lo que sería necesario hacer cambios en el software sin perder el objetivo inicial de su creación, esta refactorización podría cambiar por completo la hipótesis planteada, es importante saber cómo y cuándo es factible realizar cambios importantes, aprender a tomar decisiones informadas en la creación de Software Científico permitirá determinar el alcance del Software Científico.

- **Tiempo de desarrollo:**

El software de la industria es el resultado de un proyecto de desarrollo que dura entre 3 y 6 meses, sin embargo, cuando se trata de Software Científico se espera que el software sea desarrollado en menos tiempo, esto debido a que muchas investigaciones están pactadas para fechas específicas y cuyos resultados deberán ser presentados en determinado coloquio o congreso, el tiempo de desarrollo del Software Científico es, en algunos casos más corto que el desarrollo en la industria.

Los proyectos de investigación en donde el software es parte importante en la búsqueda de resultados o modelado de soluciones, en vital que sea construido antes de comenzar con la búsqueda de soluciones. Por lo que dependiendo de si el software será se utilice como

herramienta dentro de la investigación o siendo el resultado final de la misma, el tiempo en el que será desarrollado se verá modificado de manera importante.

- **Respaldar la investigación**

El software debe respaldar la investigación de la mejor manera, siendo el software parte importante de las investigaciones científicas, este deberá respaldar los resultados que se han obtenido, no debe haber "cajas negras" en el desarrollo de software en donde se desconozca el procedimiento interno que se está ejecutando, por lo que las pruebas sobre el software cobran una gran relevancia que permite respaldar los resultados obtenidos.

- **Claridad y correctitud**

Se debe demostrar que es lo que hizo el código de la manera más clara posible (escribiendo y documentándolo en un estilo pedagógico), ya que los usuarios finales de este no siempre tienen una formación estrictamente técnica respecto al desarrollo de software, deberá ser comprensible para cualquier investigador interesado en el tema.

- **Usuarios finales**

Al ser un software con característica tan particulares, no será usada por cualquier persona, debido a la base de conocimientos necesaria tanto para operarlo como desarrollarlo.

Criterios de Calidad:

El Software Científico juega un papel importante en la obtención de resultados de una investigación, es la herramienta que permite generar modelos, obtener evidencia, realizar alguna comprobación o ajuste en la hipótesis planteada, recientemente, los científicos han tenido que retractarse de las publicaciones debido a errores causados por fallas de software.

- Estándares: Sirven como guía a los desarrolladores cuando se trata de construir software a partir de una idea, los estándares permiten trazar un camino para lograr desarrollar un software de calidad al utilizar las normas que ya ha sido verificadas, se asegura que el sistema cuenta con la calidad adecuada.
- Herramientas: El uso de plataformas digitales para la definición y seguimiento de requerimientos o pruebas del software permite conocer el avance del desarrollo, permite llevar un mejor control sobre las actividades descritas en estándares, son útiles para cumplir las normas establecidos por estos.
- Pruebas: Las pruebas al software pueden identificar tales fallas en el código cuando son ejecutadas en etapas tempranas del desarrollo, una buena prueba de software detectara fallos o errores ya sea de lógica de programación o en el código.

Pruebas en el Software Científico

Al igual que se propone en el método científico, el desarrollo de Software Científico se basa de una hipótesis que se debe probar, por lo que se representa el proceso como un bucle donde se ejecutan pruebas y se evalúan resultados, siempre teniendo en cuenta que debe pasar por un proceso de verificación, donde la parte interesada y los desarrolladores se aseguran de que se

construye realmente lo que se quiere, por lo que se considera al desarrollo de software como un proceso en el que se debe incluir la revisión de lo que se está desarrollando.

Cuando se trata de las pruebas de software se asume que las piezas que conforman al programa han sido probadas, sin embargo, hay más de una forma de realizar pruebas, si se está probando una rutina o función con respecto a una especificación hecha (que debe cumplir con ciertas condiciones de entrada y salida), se deben considerar todos los requisitos de la especificación de entrada, sino también las acciones adicionales o no especificadas que la función hace para lograr las condiciones de salida.

El software puede probarse de manera visual, esto es rastreando el flujo de datos manualmente, o ejecutando el programa con datos de prueba y mediante ejecución de pruebas en paralelo. En la ejecución de pruebas de software la facilidad de uso y la claridad son muy importantes por lo que se deben considerar revisar los siguientes puntos (Hinsen, 2013).

- Mantener pequeño el programa, tener los resultados a un nivel intermedio y accesibles garantiza la trazabilidad de los resultados científicos.

- Buscar claridad en el código antes que optimización, muchas veces se escribe código pensando inicialmente en optimizar, sin embargo, esto puede hacer que se vuelva poco claro, se debe tener un equilibrio entre claridad y rendimiento. Un enfoque útil es escribir un programa simple y claro y probarlo exhaustivamente antes de optimizarlo.

- Ejemplos para ejecutar el software, archivos de entrada e instrucciones que indiquen cómo debe ser ejecutado, es más fácil cambiar un parámetro en un archivo de entrada existente que escribir un archivo de entrada en una sintaxis desconocida desde cero. La idea es reducir la barrera inicial para usar el software.

- Identificar dónde se encuentran los cuellos de botella, y optimizar solamente esas partes del código dejando el código original y en su lugar, colocar comentarios para ayudar a los lectores a comprender lo que hace.

- Evaluar las dependencias, que pueden ser de los lenguajes de programación, las bibliotecas, el sistema operativo. Para el desarrollo de SC se recomienda separar el núcleo computacional del programa, de la parte de la interfaz de usuario y así limitar la dependencia a algo menos relevante. En cualquier caso, se debe escribir una lista de todas las dependencias del programa y luego intentar eliminar todas las dependencias.

Validación de Software Científico

El código de software es la mejor especificación de los métodos con los cuales se resuelve un problema, y es el equivalente computacional de las fórmulas matemáticas, se debe demostrar lo que el código hace de la manera más clara posible (escribir software en un estilo pedagógico). El SC inicia como una herramienta escrita rápidamente para probar una idea

(creación rápida de prototipos), posterior a esto se realiza la optimización de rendimiento computacional.

Dentro de cualquier investigación científica se deben realizar validaciones de lo que se está desarrollando, pongamos como ejemplo las vacunas, además de las pruebas en laboratorio que muestran que la vacuna es efectiva a nivel celular, también debe ser validada por entidades especializadas o por un laboratorio externo.

En el caso del software la prueba se realiza verificando los requerimientos planteados en el proyecto, verificando que el software hace lo que se plantea como requerimiento funcional o no funcional.

En la validación, el Software Científico no se puede tratar como una caja negra, por el contrario, se deben conocer las funciones internas que permiten llegar a determinado resultado a partir de una entrada específica, esta validación permite conocer el funcionamiento del software a nivel código y a partir de esto determinar si los cálculos que el software realiza son confiables en la obtención de resultados para respaldar o rechazar la hipótesis planteada, esta validación se apoya de cierto tipo de pruebas (Hinsen, 2013):

- Ejecutar pruebas de código: Verificación del correcto funcionamiento de las funcionalidades implementadas en el software.

- Comparación de resultados contra hipótesis y/o supuestos: Comparación entre los resultados obtenidos y los resultados esperados

- Utilice bibliotecas confiables (previamente usadas o probadas), permitirá a los lectores o árbitros confiar en los resultados obtenidos.

Documentación en el Software Científico

Los modelos de desarrollo creados para la industria no se pueden aplicar de la misma forma en el desarrollo de un Software Científico, actualmente existe poca documentación formal respecto al ciclo de vida en el desarrollo de Software Científico, generalmente se adaptan modelos de desarrollo utilizados en la industria, sin embargo, el Software Científico no puede ser tratado igual que otro software en la industria.

Al desarrollar Software Científico es necesario considerar el tiempo de desarrollo y que las características solicitadas suelen ser sumamente específicas, por lo que definir la forma en cómo se deben redactar los requerimientos facilita obtener información adecuada para la construcción de software. Uno de los retos más grandes al desarrollar software es la gestión de cambios y pruebas sobre el sistema por lo que utilizar herramientas que permitan a los desarrolladores trazar un camino que muestre como ha ido evolucionando el software debido a cambios en los requerimientos resulta ser parte importante de la documentación en el Software Científico.

Capítulo 2 Estándares y Herramientas en Ingeniería de Software

Para el Software Científico existe pocos antecedentes del uso de estándares y documentación respecto al ciclo de vida en su desarrollo, generalmente se describen recomendaciones generales para que el desarrollo de SC para que sea un proceso sistemático, disciplinado y medible.

Para el desarrollo de Software Científico no existe un estándar especifico, sin embargo, los desarrolladores de este tipo de software en particular han ido adaptando las técnicas y frameworks que se conocen dentro de la industria del software, haciendo uso de ellas para el desarrollo por ciclos, colaboración entre los miembros del equipo en actividades como Repertory Grid y Card Sorting (Maller, 2012) que permiten la identificación de áreas de mejora durante su ciclo de vida.

Algunos de los estándares que se han adaptado para el desarrollo de Software Científico son:

CCMI para desarrollo (Mellon, 2018):

El Modelo de Madurez de Capacidad Integrado (CMMI, por sus siglas en inglés), es una herramienta que busca mejorar los procesos de desarrollo de software tomando como base que la calidad de un sistema está estrechamente relacionada con el proceso de construcción y mantenimiento.

ISO 15504 de calidad del software (ISO/IEC, Information Technology — Process Assessment, 2003):

Es una norma internacional que busca establecer un método objetivo para evaluar la madurez en el proceso de desarrollo de software, lo que permitirá evaluar el desempeño durante la construcción de software.

ISO 12207 procesos de ciclo de vida de software (ISO/IEC, Software Life Cycle Processes, 2008):

Estándar enfocado en el conjunto de tareas y actividades involucradas durante el desarrollo de software, administración de recursos y verificar los productos de salida de dichas actividades, su objetivo es proporcionar una estructura común que sea comprensible para todos los involucrados en el desarrollo.

ISO 29110 procesos de software en las pequeñas organizaciones (ISO/IEC, Lifecycle profiles for Very Small Entities, 2011):

Enfocado a organizaciones pequeñas desarrolladoras de software, que proporciona una guía mínima de gestión para el desarrollo de una aplicación de software hecha por un solo equipo, donde se incluyen actividades de desarrollo enfocadas a la calidad del software. La aplicación del estándar permite a las organizaciones una gestión eficaz y eficiente de los recursos de tal manera que el software desarrollado tenga la calidad suficiente para competir en el mercado global.

Esencia (OMG, 2018)

La Esencia es un estándar desarrollado por la OMG que provee elementos esenciales del desarrollo de software. Permite a los desarrolladores definir prácticas de calidad que deben llevarse a cabo para la aplicación adecuada de la Ingeniería de Software durante su creación.

El núcleo del estándar Esencia (Essence e inglés) es un conjunto de definiciones simplificado y ligero que captura la esencia y la eficacia de las prácticas de desarrollo de software, el enfoque del núcleo o kernel tiene como finalidad definir una base común para la definición de prácticas de desarrollo de software. Estas prácticas deben aplicarse de forma independiente y posteriormente es posible mezclarlas para crear un software específico. La Esencia permite aplicar tantas prácticas como se desee, de manera reutilizable y ampliable lo que significa comenzar con un método mínimo e ir agregando prácticas a medida que avanza el esfuerzo.

Este núcleo se organiza en tres áreas de interés, cada una de éstas se enfoca en un aspecto de la Ingeniería de Software, estas son Cliente, Solución y Esfuerzo.

- Esfuerzo: esta área contiene todo lo que se debe hacer en el equipo, y la forma en cómo se debe realizar el trabajo
- Cliente: Esta área de interés contiene todo lo correspondiente al uso real de los involucrados en el esfuerzo y el uso posterior del sistema de software que será desarrollado.
- Solución: En esta área contiene todo lo relacionado con la especificación y desarrollo del sistema.

Cada una de estas áreas está compuesta por una serie de *alfas* que capturan los conceptos clave involucrados en el desarrollo, permiten evaluar, rastrear y conocer el progreso de cada esfuerzo en el desarrollo. Cada alfa tiene un conjunto de *estados* predefinidos que se utilizan para evaluar el progreso y la salud del sistema de software.

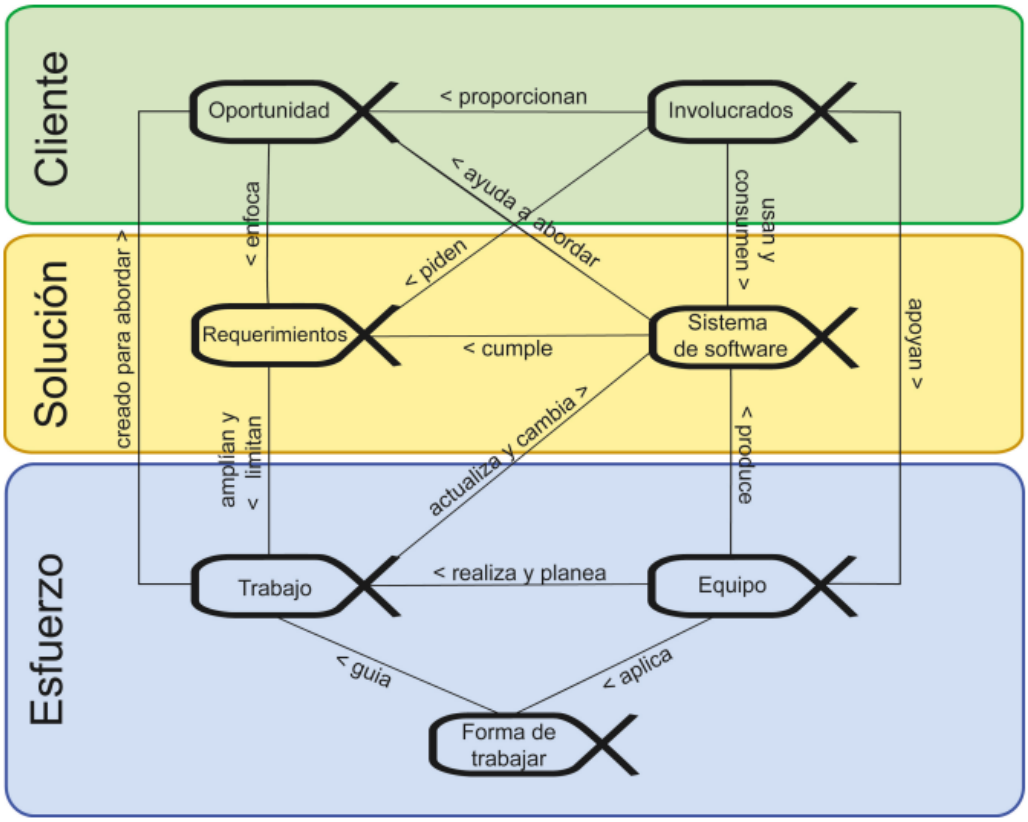


Fig. 2.0 Alfas del núcleo de la Esencia

Las alfas del área del Cliente son:

Oportunidad: son el conjunto de circunstancias que hacen apropiado el desarrollar o cambiar un sistema de software.

Involucrados: son las personas, grupos u organizaciones quienes afectan o son afectados por un sistema de software.

Las alfas del área de la Solución son:

Requerimientos: lo que el sistema de software debe hacer para afrontar la oportunidad y satisfacer las necesidades de los involucrados.

Sistema de software: es el producto principal de cualquier esfuerzo de ingeniería de software.

Las alfas del área del Esfuerzo son:

Trabajo: Actividades que implican realizar esfuerzo mental o físico con el fin de cumplir un objetivo.

Equipo: grupo de personas que participan de manera activa en el desarrollo.

Forma de trabajo: las herramientas y prácticas adaptadas utilizadas por el equipo para guiar su desarrollo.

La esencia es un estándar que se enfoca en la simplificación pues captura lo esencial de lo que es efectivo y escalable de ingeniería de software en una práctica independiente, el estándar está enfocado en establecer una base general en donde se puedan definir las prácticas de desarrollo de software, una vez que estas prácticas son definidas pueden mezclarse y adaptarse para crear métodos específicos que se ajusten a las necesidades de cada grupo de involucrados y desarrolladores. Esta flexibilidad en el estándar es una de las razones por las que se decidió utilizar a la Esencia en el desarrollo de este trabajo de investigación, ya que las áreas de interés del núcleo son sencillas de comprender.

Capítulo 3. Exploración del uso de la Ingeniería de Software en el desarrollo de Software Científico

Con la finalidad de conocer que prácticas de Ingeniería de Software se llevan a cabo actualmente en el desarrollo de software que forma parte de las investigaciones científicas, se llevó a cabo un cuestionario que permita tener un diagnóstico para ser contestado por alumnos del posgrado de Ciencias e Ingeniería en Computación cuyas investigaciones han requerido el desarrollo de software científico especializado, en el anexo 1 llamado Cuestionario de exploración de este documento se puede encontrar el cuestionario propuesto.

Las preguntas fueron pensadas para ser respondidas de manera abierta permitiendo a los alumnos ahondar más en la forma en que se apoyan de la Ingeniería de Software para desarrollar sus proyectos de maestría, las respuestas a dicho cuestionario fueron proporcionadas por cinco alumnos del posgrado se enlistan a continuación:

¿Define en tus palabras que es el Software Científico?

1. Software cuyo objetivo es ayudar a probar o refutar una hipótesis científica
2. Un sistema experimental, controlado, en donde el programador puede modificar el sistema dependiendo de las necesidades con el objetivo de medir un conjunto de variables que permitan sustentar sus hipótesis
3. Software innovador cuyo propósito no necesariamente es el de ganar dinero.
4. Software usado para realizar una labor científica
5. software para generar resultados de la investigación o aplicar los algoritmos diseñados

Los alumnos identifican algunas características del Software Científico tales como que es un sistema en constante cambio y permite sustentar las hipótesis sobre las que se trabaja en una investigación científica.

¿Cuáles crees que serían las diferencias entre el Software Científico y el software comercial?

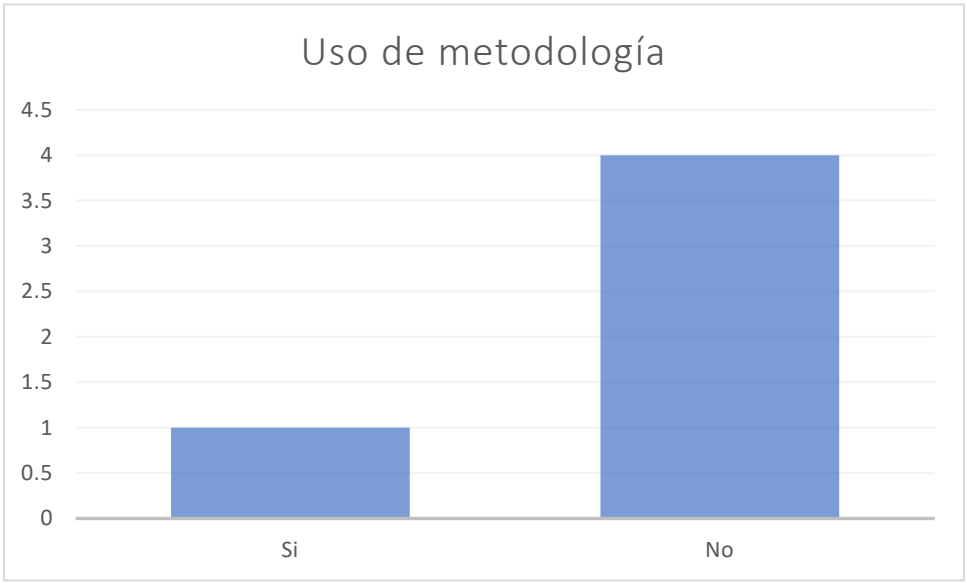
1. El objetivo, Uno agrega valor a un cliente y otro responde una pregunta científica.
2. El software comercial generalmente no se enfoca en un estudio sobre un área en particular, más bien, está dirigido al usuario final y suele permitir al usuario más libertad
3. Pienso que el científico tiende más a la innovación que a generar dinero.

4. En mi opinión son conceptos ortogonales. El Software Científico enfatiza a los usos del software por parte del usuario (realizar una labor científica). El software comercial se refiere a la relación entre los proveedores del software y los usuarios del software, en donde el Énfasis está en la venta o renta del producto. Hay Software Científico comercial, Software Científico no comercial y software comercial no científico
5. las metas, la organización y el presupuesto

Los alumnos reconocen que existen diferencias entre el software desarrollado de manera comercial y el Software Científico, mientras el software comercial tiene múltiples usuarios finales, así como el objetivo de obtener ganancias monetarias por dicho software. El Software Científico ayuda a responder preguntas de estudio concretas.

¿Actualmente sigues alguna metodología de Ingeniería de Software en el desarrollo de tu software?

1. No
2. Si, metodología de prototipos
3. No.
4. No
5. No



La mayoría de los alumnos que respondieron al cuestionario no están utilizando ningún tipo de metodología, sin embargo, sabemos que la Ingeniería de Software proporciona soporte operacional, mantenimiento y calidad en la creación de un sistema.

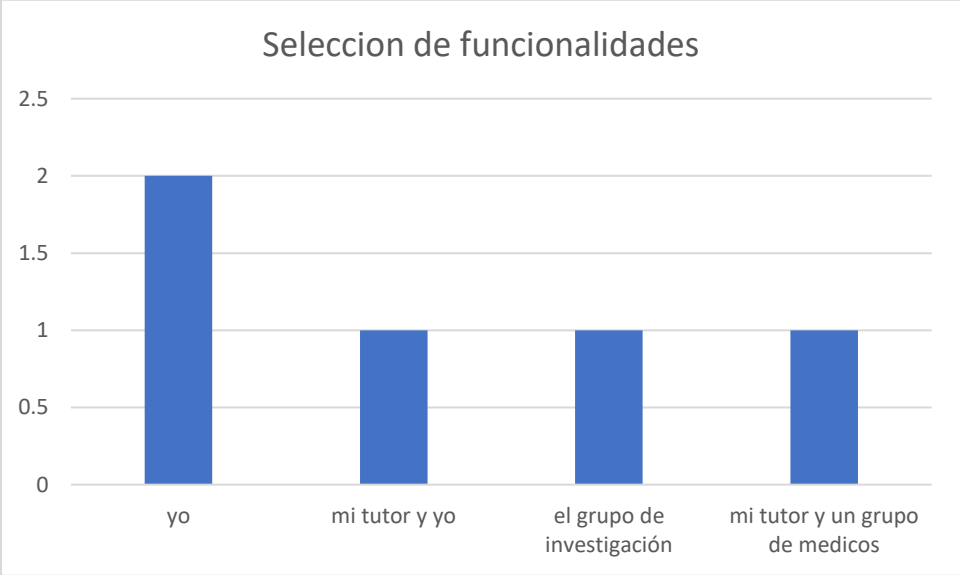
¿Cómo obtienes los requerimientos de tu software?

1. Busco que cumpla con confirmar o refutar la hipótesis. Además de eso solo sigo buenas prácticas generales para el desarrollo.
2. Con base a variables del hardware, y al análisis de las variables a medir en cuestionarios de experiencia de usuario
3. Mi tutor y un grupo de médicos me los dicen (es un simulador quirúrgico).
4. Los problemas que el grupo de investigación intenta resolver y la exploración de posibles soluciones determina los requerimientos del software. En nuestro caso particular el software que implementamos es tanto una herramienta para realizar nuestra investigación como el objeto de estudio, por lo que los requerimientos se van especificando de forma iterativa y suelen cambiar junto con el desarrollo del software
5. en lo desarrollado en la maestría están dados por la investigación que yo mismo realizo para implementar las ecuaciones utilizadas y aplicar a los datos que tengo

Los requerimientos son un tema sumamente extenso dentro del desarrollo de software, existen muchas herramientas que permiten a los desarrolladores redactar requerimientos de forma clara y que permiten visualizar posibles cambios o mejoras dentro del sistema.

¿Quién toma la decisión de que funcionalidades son importantes en tu software?

1. Para mi tesis mi tutor y yo
2. Yo
3. Mi tutor y un grupo de médicos.
4. El grupo de investigación
5. Yo



En cuanto a la toma de decisiones sobre qué características debe cumplir el software que se planea desarrollar hay pocos involucrados, por lo que la definición de dichas funcionalidades debe ser descrito de manera clara para que todos los involucrados entiendan los procesos que deban ejecutarse.

¿Evalúas la calidad de tu software?

1. De forma general. No sigo una metodología establecida.
2. Si
3. Si
4. Algunas partes del software son verificadas formalmente, otras no. No usamos una métrica para evaluar la calidad del software.
5. Si

Verificar que el software sea desarrollado basado en estándares de calidad y que cumplan con las características esperadas por los desarrolladores es sumamente importante dentro del ámbito científico, puesto que el software debe ser revisado por expertos que determinen si los procesos que ejecuta son los adecuados para resolver el problema, así como aceptar o rechazar la hipótesis planteada dentro de la investigación.

¿Con que tipo de pruebas?

1. Tiempo de ejecución, modularidad, robustez, control de excepciones, facilidad de interacción con el usuario
2. Con base a artículos que respalden la efectividad de los métodos utilizados
3. Cuestionarios a los médicos.
4. Partes del software tienen pruebas formales que verifican que es correcto. Usamos algunas unitarias para implementaciones de interfaces genéricas. Finalmente hacemos pruebas de estrés y de rendimiento, pero estas no las tenemos automatizadas, si no que diseñamos pruebas antes del uso del software para un problema particular
5. análisis del algoritmo y comparación de resultados pequeños con datos previamente seleccionados y resultados hechos a mano o con otra herramienta como Excel

La ejecución de pruebas sobre el software en etapas tempranas reduce la probabilidad de encontrar errores graves más adelante o incluso durante la entrega del software, su importancia es fundamental tratándose de software que debe ser usado para obtener respuestas a preguntas concretas y garantizar que los resultados son confiables.

¿Cómo llevas la documentación de tu desarrollo?

1. En archivos readme principalmente
2. Generalmente esquematizo los módulos y suelo usar diagramas de flujos para darme entender y a los siguientes programadores
3. Con comentarios en mi código.
4. Una vez que decidimos que un módulo del software debe preservarse, lo escribimos en forma de programación literaria. Sin embargo, la documentación de su uso está muy ligada a su implementación. No escribimos manuales de usuario y el registro de los cambios de las funcionalidades queda relegada al historial de commits en git
5. en comentarios dentro del código y un documento que describe cómo funciona y como se relaciona con la parte teórica dentro de la tesis

La documentación del desarrollo, el uso y los resultados obtenidos resulta útil al respaldar una investigación científica que tiene como pieza clave el uso de software especialmente desarrollado para probar la hipótesis propuesta.

Con base en la información obtenida a partir de las respuestas de los alumnos, el objetivo de este trabajo será generar una propuesta de directrices para el desarrollo de Software Científico, a partir del conjunto de prácticas de desarrollo de software descritas en el Estándar Essence (OMG, 2014), permitiendo a quienes desarrollan Software Científico tener acceso a herramientas o normativas de la Ingeniería de Software que puedan aplicarse de manera sencilla durante el ciclo de vida de este tipo del software.

Capítulo 4: Propuestas de directrices para el desarrollo de Software Científico. Área de Cliente

El estándar conocido como Esencia provee un conjunto de definiciones y prácticas comunes que incluyen los elementos esenciales dentro del desarrollo de software a los que nombra ***núcleo***, estos elementos tienen estados que permiten conocer el progreso y la salud del sistema, cada esfuerzo realizado por los involucrados lleva a un cambio de estado. Dentro del núcleo se organizan diversas áreas de interés cada una con un grupo de Alfas que son representaciones de las cosas más importantes con las que se trabaja durante la creación de un software, tales como: Oportunidad, Involucrados, Requerimientos, Sistema de software

Las directrices propuestas favorecen la documentación del proyecto de desarrollo de software, ya que están diseñadas de acuerdo con diferentes estándares internacionales. De forma complementaria a las directrices se han diseñado varias plantillas de apoyo con la finalidad de docume3ntar la toma de decisiones importantes, estas plantillas se encuentran al final de cada una de las directrices propuestas.

La Esencia establece áreas de trabajo llamadas *Cliente, Solución y Esfuerzo*, cada una de estas áreas contienen *alfas*: en estas se busca identificar la oportunidad de desarrollo de un software; conocer a los involucrados; apoyar la definición y redacción de los requerimientos; el modelado del software, su construcción y prueba, finalmente su mantenimiento y evolución.

Conceptos teóricos

Desarrollo Conjunto de Aplicaciones (JAD)

Enfocada en la obtención de requerimientos donde los involucrados (investigadores, programadores y usuarios) trabajan en conjunto para construir software, el desarrollo conjunto de aplicaciones permite que todo el equipo participe durante la obtención de requisitos permitiendo a los involucrados interactuar con el sistema conociendo su funcionamiento deseado.

1. Identificar objetivos y limitaciones: establecer las expectativas de los involucrados, en este paso se intenta evaluar tanto el diseño como la complejidad de lo que se quiere implementar con la finalidad de dimensionar todo dentro del proyecto.
2. Identificar factores críticos de éxito: identificar los factores críticos de éxito para el desarrollo evitar divagar sobre los cambios y extras en la creación de un software, planificar los resultados permite juzgar la eficacia y calidad del software-
3. Definir los entregables del proyecto: los entregables son la documentación y un diseño, es importante definir el nivel de detalle y el cómo escribir la documentación de las entregas.
4. Definir cronograma de actividades: en los primeros días de desarrollo el equipo busca adaptarse al nuevo entorno y conocerse, pero se adaptarán poco a poco el equipo buscara realizar de manera más rápida y eficiente las actividades programadas.
5. Seleccione los participantes: definir a quienes son los usuarios y expertos involucrados permite impulsar cambios para la mejora del desarrollo

6. Preparar el material: esto incluye la documentación a presentar, diagramas, software etc.
7. Organizar las actividades y los ejercicios: el facilitador se encarga de diagramar las actividades y quienes las van a realizar, como diagramas que expliquen cómo funcionan etc.
8. Preparar, informar educar a los participantes: los participantes deben ser conscientes de los objetivos y limitaciones del proyecto y de los resultados esperados, se debe proporcionar una definición clara del alcance del proyecto para todos los participantes.
9. Coordinar la logística: esto incluye proyectores, el lugar de reunión, pc, mesas, marcadores etc. La distribución del espacio debe favorecer la interacción entre los participantes

Oportunidad

La Oportunidad es un alfa de la Esencia son las condiciones que permiten el desarrollo o creación de un software (OMG, 2018), justifica la creación del software o su modificación. En esta alfa el equipo comparte la visión de cuáles son las necesidades de los interesados del por *qué* y *cómo* un software podría resolverlas, dicha oportunidad permite generar los primeros requerimientos.

El alfa Oportunidad busca establecer la necesidad del software, identificando las necesidades de los involucrados, los retos en la construcción del software y la viabilidad del proyecto. Las alfas Oportunidad e Involucrados están estrechamente ligadas, ya que los involucrados son quienes conocen lo que necesitan que haga el software, determinan sí es viable y trabajan en conjunto para su desarrollo. Por su parte, cada involucrado aporta, al desempeñar las responsabilidades de su rol asignado, el avance en la construcción de un software de calidad capaz de resolver las necesidades.

Para identificar la Oportunidad se partirá de una hipótesis o de la necesidad de tener una herramienta capaz de automatizar y resolver tareas dentro de una investigación. La herramienta que se propone para identificar esa oportunidad es un cuestionario, que consiste en un conjunto de preguntas, preparado sistemáticamente para obtener datos que interesan en la investigación, y para conocer más acerca de la población con la que se trabaja, dichas preguntas son formuladas con claridad suficiente para que, a partir de la respuesta maximizar la probabilidad de obtener información que pueda ser útil.

A través de contestar preguntas se establecerá el objetivo u oportunidad, tomando en cuenta que las respuestas a estas preguntas deben ser la razón de la creación o modificación del software a desarrollar, también permitirá conocer el proceso que para comprobar y obtener los resultados de la hipótesis planteada.

- ¿Qué?
- ¿Cómo?
- ¿Para qué?
- ¿Resultado?

Definir el enfoque del software que será desarrollado permite a los investigadores tener claro el producto final que se quiere construir, y las características que este debe poseer, minimizando la ambigüedad a la hora de comunicar la idea a los programadores o al querer definir los requisitos que dicho software debe cumplir.

El cuestionario presentado a continuación fue creado como propuesta para que los involucrados establezcan una serie de ideas deseadas dentro del software, dándoles un objetivo y expresando la oportunidad porque es importante considerarlas antes del sistema que se planea desarrollar.

Cuestionario para la obtención del Objetivo u Oportunidad			
¿Que?	*¿Como?*	*¿Para qué?*	*¿Resultado?*
Funcionalidades u operaciones específicas que el software debe realizar	Características destacables en el software	Importancia de las funcionalidades esperadas del software	Valor esperado después de ejecutar las operaciones y las características del resultado exitoso

El objetivo de contestar estas preguntas es conocer las circunstancias apropiadas para que el software sea desarrollado, establecer las características, operaciones, procesos y resultados que deben obtenerse a partir de este software dará paso a que el equipo comparta la idea final del software, y comenzar a generar ideas de cómo construirlo.

Diferentes involucrados verán la oportunidad de diferente manera, y esperarán resultados diferentes de cualquier sistema de software a producirse para hacerle frente a la oportunidad, los involucrados proporcionan los requerimientos y financiación para el sistema. Después de definir los requerimientos es de suma importancia asegurar que los involucrados tengan una comprensión compartida sobre la oportunidad y la razón de los requerimientos, este acuerdo da pie a la justificación para el desarrollo.

Involucrados

Los involucrados según la Esencia, en un proyecto de desarrollo son en su mayoría el grupo de personas quienes afectan y serán afectados por el software, los involucrados proporcionan la oportunidad, los requerimientos y son quienes proveen los recursos para que se lleve a cabo el proyecto. El esfuerzo conjunto de todos los involucrados es necesario para lograr con éxito la creación de software por lo que es necesario un fuerte compromiso de inicio a fin del desarrollo.

Identificar a los involucrados permite también conocer las responsabilidades de cada uno, son quienes afectan en mayor o menor medida el desarrollo de software, por lo que deben ser

conscientes de que deberán estar involucrados y comprometidos con el software. Los involucrados deben trabajar junto con los programadores para lograr el resultado esperado.

- ¿Rol dentro del proyecto?
- ¿Área de conocimiento?
- ¿Actividades en las que participa?
- ¿Qué tipo de dudas resuelve?
- ¿Quién/es?

En el Software Científico los involucrados en la creación de un software son principalmente los investigadores a cargo del estudio científico por lo que su formación puede centrarse en un área diferente a programación, esto no significa que su participación en el desarrollo sea limitada o menos relevante, por el contrario, permite tener diferentes puntos de vista para atacar de una forma más eficaz el problema o necesidad.

Se proponen las siguientes definiciones para identificar los roles que desempeñan los participantes en un proyecto de desarrollo de SC a los que llamamos *Involucrados*, así como las actividades que deberán realizar durante el proyecto y las características destacables de cada rol. Seleccionar a un involucrado para que asuma determinado rol dentro del desarrollo es importante pues de esta forma se identifica quién puede tomar ciertas decisiones o quién posee el conocimiento y experiencia para resolver problemas que puedan surgir.

Para el alfa de los Involucrados se busca identificar a las personas que participarán en el proyecto. Se propone este cuestionario para ayudar a identificarlos, cada participante deberá contestarlo y luego se consensa entre todo el equipo. En algunos casos dependiendo del número de involucrados, habrá roles que deban ser ocupados por una misma persona y una persona puede ocupar varios roles.

Cuestionario para identificar a los involucrados en el desarrollo				
¿Rol dentro del proyecto?	*¿Quién?*	*¿Área de conocimiento?*	*¿Actividades en las que participa?*	*¿Qué tipo de dudas resuelve?*
Que **rol** es el más adecuado de acuerdo con su experiencia o nivel de conocimiento	Identificar la persona que de acuerdo con el equipo es la mejor calificada para **desempeñar** dicho rol	**Formación** profesional	Dependiendo del Rol cuales son las **responsabilidades** que tiene	**Área de experiencia** / cargo dentro de la investigación u organización

Algunos marcos de trabajo enfocados en el desarrollo de software trabajan con varios roles predefinidos, sin embargo, para el desarrollo de Software Científico los interesados y los roles en el desarrollo cambian, pues algunas veces la persona que codifica el programa y quien provee los recursos son la misma persona. Para definir los roles de los involucrados en el desarrollo de un software se identifican primero a los involucrados, lo que aportaran dentro del proyecto y las necesidades técnicas del mismo.

En el desarrollo de Software Científico los roles propuestos son los siguientes:

Investigadores

Al tratarse de la construcción de un software especializado son necesarios conocimientos muy particulares para generar una solución basada en software. Los procedimientos que se deben seguir para la solución de un problema son muchas veces resolver sistemas de ecuaciones que rigen el comportamiento de fenómenos de la física o comportamientos en la naturaleza. Los investigadores son quienes conocen el método de solución al problema planteado, por lo que deben ser capaces de comunicar a los programadores el procedimiento de solución de forma clara y precisa, traducen del lenguaje matemático a uno que los programadores puedan codificar.

Habilidades ideales:

- Empatía
- Trabajo en red
- Analista objetivo
- Buena comunicación verbal y escrita

Responsabilidades:

- ✓ Proponer las funcionalidades de la solución
- ✓ Verificar la corrección de las funciones de software
- ✓ Resuelve las dudas que surgen respecto a la solución

Programadores

Los involucrados con este rol son quienes codificarán el software hablando a nivel computacional, sus conocimientos permiten crear un conjunto de archivos escritos en algún lenguaje de programación, crean las funciones del software a partir de los requerimientos construidos por todo el equipo, el programador es quien construye los prototipos que se muestran cada cierto tiempo al resto de los involucrados, la experiencia en construcción del software puede ayudar al resto de involucrados a identificar mejor el alcance del software deseado.

Habilidades ideales:

- Analista crítico
- Ordenado y metódico

- Resolución de problemas
- Capacidad autocritica para mejorar su rendimiento
- Conocimientos técnicos de lenguajes de programación

Responsabilidades:

- ✓ Corregir defectos en el código
- ✓ Comunicar el avance del software
- ✓ Documentar el código del software
- ✓ Escribir el código que formara parte del software

Testers

Este rol es el encargado de examinar el prototipo generado por los programadores. Los involucrados que pertenecen a este rol, prueban que las operaciones se ejecuten de manera adecuada apoyándose de las descripciones hechas en los requerimientos. El tester valida y verifica los procesos del software, si se detectan errores o inconsistencias en la ejecución del software deben ser reportados a los programadores y al resto del equipo, este rol permite identificar los puntos fuertes y los puntos donde el software puede mejorar a partir de cambios, la exploración del software mejora la calidad de los resultados.

Habilidades ideales:

- Comunicación
- Capacidad de análisis
- Conocer los detalles de las pruebas de software

Responsabilidades:

- ✓ Ejecutar pruebas
- ✓ Revisar detalladamente el software
- ✓ Documentar resultados de las pruebas
- ✓ Comunicar dichos resultados y sus posibles mejoras

Cuando se trata de una investigación científica hay otro tipo de involucrados pues los recursos destinados a esta investigación deben ser administrados por algún organismo superior a quien deben entregarse informes donde se detalla cómo se han usado los recursos destinados a dicha investigación.

Administradores

Los proveedores son quienes deciden como asignar los recursos disponibles en el proyecto, comprender el porqué de crear una solución en software es importante permite tomar decisiones sobre la distribución de personal, dinero, tiempo etc. Pueden establecer criterios para la construcción y entrega de la solución, por lo que es importante que comprendan la forma en que se construye dicha solución. Los administradores pueden determinar la viabilidad del desarrollo y de la misma forma decidir si el resultado final es el esperado.

Habilidades ideales:

- Facilitación
- Negociación
- Administración

Responsabilidades:

- ✓ Entender la oportunidad
- ✓ Entender la complejidad
- ✓ Administración de todos los recursos

Es importante considerar dichos roles durante el desarrollo de Software Científico ya que facilita la interacción entre involucrados debido a lo complicado de interactuar directamente con todos los involucrados, los roles propuestos y las habilidades ideales permiten que las personas que los desempeñan tengan las competencias necesarias para mejorar la interacción de cada involucrado, lo que permite al equipo hacer un esfuerzo conjunto con la finalidad de desarrollar un software con los estándares de Ingeniería de Software adecuados.

El número de involucrados dentro de un desarrollo de Software Científico suele ser pequeño por lo que los roles considerados como indispensables para alcanzar los objetivos del software se resumen a Administradores, Investigadores, Programadores y Testers. Es importante mencionar que en algunas ocasiones las actividades o decisiones que deben ser tomadas para el avance del software no dependen de un solo rol, por lo que las intersecciones en el diagrama mostrado a continuación representan una unión entre los involucrados estas uniones pueden ir desde comunicación en avances de software, toma de decisiones, incluso los problemas que pudieran surgir al momento de crear el sistema.

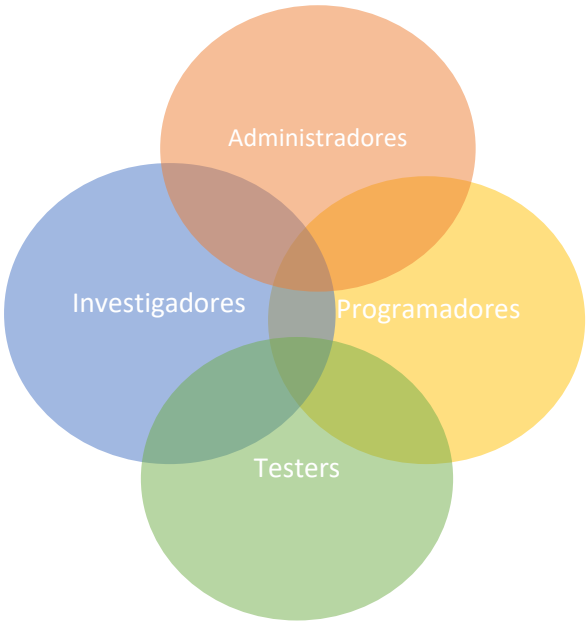


Fig. 4.1 Diagrama de Involucrados

Capítulo 5 Área de Solución

En este capítulo se explora la obtención de requerimientos en el desarrollo de software su importancia para definir las características vitales que debe cumplir el sistema desarrollado, posteriormente se describe una propuesta para la redacción de requerimientos que contengan la información necesaria para que los programadores sean capaces de codificarlos. Tomando en cuenta que el Software Científico debe ser desarrollado en cortos espacios de tiempo y bajo una hipótesis que puede estar en constantes cambios es importante saber catalogar los requerimientos propuestos por los involucrados, una de la estrategias más sencillas y útiles para seleccionar requerimientos es la técnica MoSCoW que se detalla más adelante en el capítulo.

5.1 Requerimientos

Una de las áreas más importantes y complejas de la Ingeniería de Software es la ingeniería de requerimientos encargada de identificar, describir, analizar, negociar y validar los requisitos con los que un software debe contar para funcionar.

Los requerimientos son las características vitales del software no importa si son 10 o 100, deben estar perfectamente descritas y definidas, saber quién es el responsable, como se va a construir en términos técnicos y delimitados. Para identificar y escribir de forma precisa los requerimientos del software es necesario tener claras las operaciones o acciones que deben realizarse con la finalidad de obtener un resultado satisfactorio al ejecutar el software.

Hay una gran variedad de técnicas de obtención de requerimientos, sin embargo, la técnica más apropiada dependerá del proyecto que se está desarrollando, debido a la naturaleza del Software Científico que busca generar software útil en un espacio de tiempo muy corto algunas de las siguientes técnicas resultan adecuadas para la obtención de requerimientos.

5.1 Herramientas para la obtención de Requerimientos

Las herramientas de obtención de requerimientos son sumamente útiles y sencillas de usar, proporcionan a los desarrolladores una herramienta de descripción de que es lo que se debe codificar, mientras más claros y organizados se encuentren los requerimientos será más sencillo para el programador saber qué debe codificar y cómo debe hacerlo. Construir prototipos permite tener un avance tangible del sistema e identificar si es necesario hacer cambios para guiar el proyecto en el camino correcto.

Algunas de las herramientas utilizadas para la redacción de requerimientos son las historias de usuario que permiten construir los requerimientos, a través de responder preguntas sobre qué es lo que se desea ver en el software y para qué, esto resulta obvio cuando se trata de describir características deseables en un software, la parte que más llama la atención en la estructura de una historia de usuario es la sección conocida en algunos marcos de desarrollo ágil como

"*Definition of Done*" la cual permite hacer una validación para concluir si el requerimiento ha sido concluido de manera satisfactoria.

Una historia de usuario se escribe describiendo una característica deseable por parte de algún rol determinado dentro del proyecto y a la finalidad de dicha característica, se añade también la definición de terminado, que permitirá validad que dicho requerimiento se ha completado de manera satisfactoria.

A continuación, se muestra un ejemplo de historia de usuario:

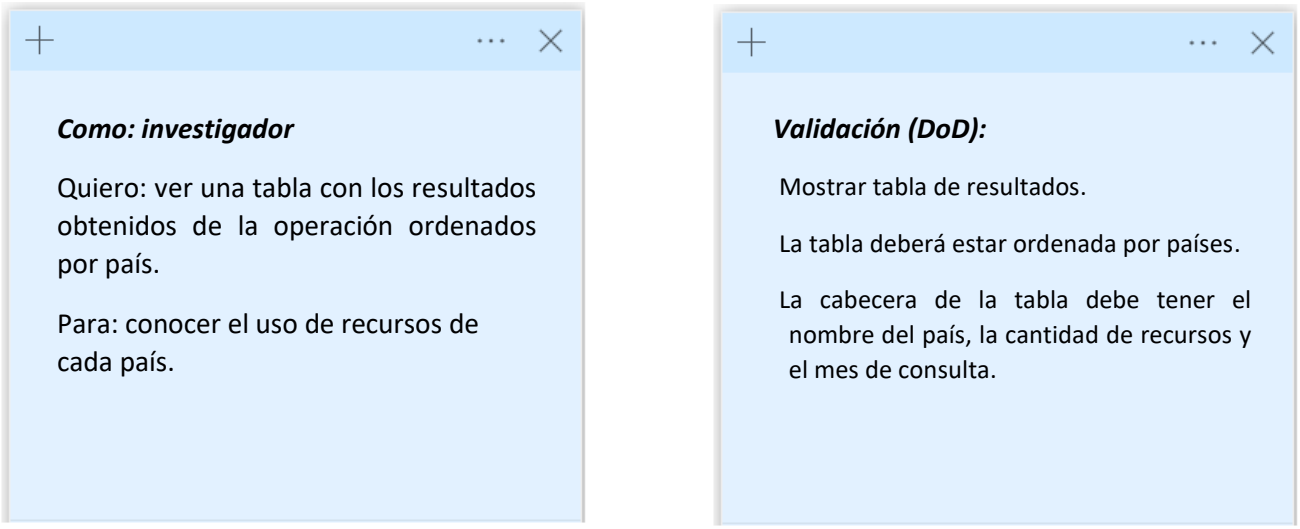


Fig. 5.1 Historia de usuario

Escribir requerimientos resulta una tarea compleja y que requiere de la mayor claridad por parte de los involucrados. Es importante, al llevar a cabo la actividad donde se obtendrán los requerimientos, se encuentren presentes todos los involucrados de esta forma los requerimientos serán construidos a partir de diferentes puntos de vista, dando claridad y completitud a su definición.

Algunas de las características más útiles a la hora de definir requerimientos en el Software Científico son la validación, la descripción del proceso que requiere ejecutar el requerimiento, la priorización de requerimientos también es un punto importante que considerar en el Software Científico, donde se deben seleccionar los requerimientos que permiten obtener los resultados durante las primeras versiones del software. En la obtención de requerimientos para el Software Científico se debe tener especial cuidado al describir el proceso que debe realizar dicho requerimiento y su posterior verificación durante la etapa de pruebas.

Validación

Esta sección dentro del requerimiento de software especifica las características que debe cumplir un requerimiento cuando se ejecute una prueba sobre éste o para ser considerado como completado, esta validación puede ser escrita en forma de lista de verificación o *check list* ya que no necesita ser en extremo detallada a diferencia de las pruebas sobre el software.

La lista de verificación específica para cada requerimiento debe contemplar la consistencia, con la finalidad de evitar contradicciones entre el requisito y el resultado presentado. La completitud es donde se verifica que el requisito ejecute todas las operaciones y funciones descritas. Escribir los puntos para validación como una lista de resultados esperados coherente y clara, permitirá realizar una validación de que se cumpla el requerimiento de manera adecuada.

Priorización

Es importante catalogar los requerimientos pues de esta forma se podrán seleccionar aquellos que, al construirse, permitan ver un prototipo del software pequeño pero que sea capaz de entregar resultados. La toma de decisión sobre la prioridad que tiene cada requerimiento depende en gran manera de que porciones de software son más importantes, si los involucrados consideran que realizar consultas a las bases de datos tiene mayor prioridad que obtener la suma de 2 elementos, entonces la creación de los requerimientos involucrados en la consulta a bases de datos deberá tener prioridad a la hora de codificar el software.

En el Software Científico se puede realizar la priorización utilizando diferentes métodos y herramientas, dichas herramientas permiten a los involucrados discernir entre todos los requerimientos descritos y seleccionar cuales requieren una mayor atención o cuales necesitan ser desarrolladas en etapas tempranas. Priorizar requerimientos permite al equipo organizar el trabajo y saber por dónde comenzar a construir el software, acordar entre todos los involucrados cuales deben ser las funcionalidades prioritarias permitirá al equipo trabajar con un objetivo en común y ver resultados de manera temprana.

A continuación, se presenta una herramienta útil en la priorización de requerimientos, cabe destacar que es posible categorizar requerimientos de diferentes ámbitos por ejemplo algunos relacionados con la interfaz o con operaciones de consulta etc. Todos los requerimientos que se consideren prioritarios deben estar relacionados entre sí de alguna forma, esto permite una transición entre la selección de requerimientos, la construcción y la verificación junto con pruebas.

El método MoSCoW

Es una técnica desarrollada por Dai Clegg (Dai Clegg, 1994), que sirve para determinar la priorización dentro de los proyectos de desarrollo, el equipo debe tener claro que el objetivo de priorizar no es añadir todas las funcionalidades posibles y ningún involucrado tiene prioridad sobre el resto del equipo, por el contrario, todos deben estar de acuerdo con la selección de los requerimientos sobre los que se va a trabajar de forma inmediata.

Para realizar una priorización de forma estratégica y ordenada el método *MoSCoW* utiliza cuatro categorías: *Debe tener (Must have), debería incluir (should have), podría incluir (could have), no se van a hacer (won't have).*

Must have (Debe tener)

Dentro de esta sección se encuentran las características esenciales del software, son aquellas donde el proyecto carece de sentido, para determinar si un requerimiento pertenece a esta categoría se deben hacer las siguientes preguntas.

- ¿Qué pasaría si se entrega el software sin esta funcionalidad?
- ¿El software funcionara sin esta característica?

Should have (Debería incluir)

En esta categoría se encuentran los aspectos críticos del proyecto, pero no son imprescindibles, pueden ser considerados como funciones secundarias pero importantes para complementar la ejecución del sistema.

- ¿Esto puede esperar hasta el próximo lanzamiento?
- ¿Cuánto valor añade esta característica al proyecto?

Could have (Podría incluir)

Los requerimientos de esta categoría pueden verse como lista de deseos son aquellas que estaría bien tenerlas, y que añadirían algún valor al proyecto.

- ¿Aporta algún valor al sistema?
- ¿Es necesario para ejecutar alguna función básica?

Won’t have (No se van a hacer)

Estas características no aportan ningún beneficio actualmente al proyecto y podrían, o no, ser consideradas después.

- ¿Hay alguna posibilidad de que esto tenga importancia en el futuro?
- ¿Sin esta característica el sistema funciona sin problemas?

Cuando se trata de construir requerimientos para un sistema es más sencillo tener una estructura definida de cómo es un requerimiento completo, en otras palabras, una plantilla resultaría útil al momento de redactar los requerimientos. Al redactar requerimientos es importante considerar que deben centrarse en objetivos realistas y ser implementados con la tecnología y recursos disponibles, los requerimientos deben contar con alguna forma de comprobar que cada requisito en ellos se cumple.

5.3 Directrices para la definición de los requerimientos

En el establecimiento de los requerimientos es importante que las características indispensables del software ya han sido acordadas y aceptadas por los involucrados. A continuación, se describe una lista de pasos que se deben seguir para lograr una adecuada redacción de requerimientos.

- Identificar las funcionalidades más destacables con las que debe contar el software.

- Escribir una narrativa de lo que los involucrados desean conseguir construyendo esta funcionalidad y cuál es el proceso que debe realizar para lograr el objetivo. Es una *descripción* breve de que se espera ver en el software.

- Escribir las características con las que debe cumplir el requerimiento al momento de realizar pruebas. La definición de terminado o *Definition of Done* son los criterios de aceptación que se deben cumplir al momento de probar el software, puede ser escrita como una lista, de esta forma realizar una revisión se convertirá en una tarea sencilla.

- Una vez que la Descripción y los criterios de aceptación han sido redactados, es importante identificar quien ha propuesto el requerimiento y quien se encuentra a cargo de codificarlo. En este paso se eligen al *involucrado* que ha propuesto y definido el requerimiento y el *responsable* quien se encargara de construir el requerimiento.

- Cuando el requerimiento ha sido redactado y todos los involucrados comprenden el proceso que debe ejecutar, como debe ser evaluado y determinar si está completo, es momento de asignarle un nivel de prioridad, es decir, si los involucrados consideran que un requerimiento es indispensable dentro del software entonces su prioridad debe ser alta este proceso puede ser realizado utilizando el método MoSCoW, añadiendo los requerimientos en cada una de las categorías de MoSCoW.

- El método MoSCoW se basa en un acrónimo *Mo por Must Have (debe tener)* donde se añaden las tareas indispensables, que deben llevarse en primer lugar. *S de Should Have (debería tener)* categoría que agrupa a las tareas que aporta un valor añadido al proyecto, pero pueden diferirse con el tiempo. *Co de Could Have (podría tener)* tareas que pueden ser realizada si hay tiempo son un extra. *W de Won´t have (no tendrá)* esta sección hace referencia a las tareas que no son de gran valor para el desarrollo, puede

ser un requerimiento que se realice en algún futuro cuando todas las tareas ya han sido completadas.

- Finalmente asignar un nombre al requerimiento redactado, un nombre que sea sencillo de recordar y que ayude a los involucrados a identificar de que trata un requerimiento en particular, asignar un nombre al requerimiento.

Como ayuda se propone construir una tabla como la siguiente para cada requerimiento a fin de tenerlos juntos y documentados como apoyo al resto del desarrollo.

Fig. 5.2 Plantilla para redacción de requerimientos

Capítulo 6. Sistema de Software

Los resultados en las investigaciones científicas deben ser confiables y claros, por lo tanto, el software que se desarrolla para obtener dichos resultados debe ser robusto y confiable, de lo contrario la credibilidad de los datos obtenidos se verá reducida. Un diseño construido con las técnicas adecuadas puede asegurar que el software tenga un funcionamiento adecuado frente a diversas circunstancias y de la misma forma que sea eficiente, aprovechando los recursos a su disposición.

El sistema de software es una alfa, que forma parte del área de Solución de la Esencia. Esta alfa incluye tres actividades que son: modelar la estructura del software, construirlo y probarlo, que revisaremos en este capítulo.

En esta alfa los requerimientos redactados por los involucrados, la administración de recursos y consideraciones técnicas trabajadas en las otras alfas se alinean con la finalidad de definir una representación o modelo de producto o sistema de software.

El diseño o modelado tiene como objetivo identificar y agrupar la estructura del software, así como sus características y conceptos que permitirán el desarrollo de un producto de calidad y establece ciertas reglas para el sistema de software que se desea crear.

El diseño permite evaluar características de lo que se desea a construir, proporciona detalles de los componentes y cómo se relacionan. A través del diseño se pueden realizar pruebas y mejoras antes de que los programadores se encarguen de escribir el código. Un buen diseño de sistema debe representar la arquitectura, interfaces, componentes, conocer los subsistemas y cómo se comunican entre ellos.

6.1 Diseño o modelado del sistema de alto nivel

Durante el diseño de alto nivel (M. del Carmen Gómez, 2019) se determinan los subsistemas que conforman el sistema de software principal, éstos pueden descomponerse en módulos y submódulos, al proceso de descomponer el sistema en módulos más básicos se le conoce como modularización. El proceso contrario es el ensamblaje, permite conocer las dependencias entre módulos y cómo deberán comunicarse para que el sistema haga lo que se espera. Tanto la modularización como en ensamblaje permiten a los desarrolladores tener una idea clara de que deben desarrollar y la forma en que estos módulos deberán comunicarse entre sí.

Este tipo de diseño tiene como objetivos los siguientes puntos

1. Definición de la arquitectura
2. Definición de la estructura de datos para cada módulo
3. Definición de interfaces para el usuario y entre módulos
4. Propuesta de solución que incluya también los requerimientos no funcionales (fiabilidad, seguridad, eficiencia etc.)

6.1.1 Diagramas de paquetes y su modelado UML

Un diagrama de paquete es un diagrama compuesto únicamente de paquetes y sus dependencias o formas de comunicación entre ellos. Permite visualizar los aspectos generales del sistema. Un paquete es una construcción gráfica, diseño o modelo que permite organizar los elementos del sistema, los paquetes del diagrama son los módulos o funciones más importantes dentro del sistema y son representados con el símbolo de carpetas de archivos.

Los paquetes en un diagrama agrupan y organizan los elementos que componen el sistema desarrollado y sus funcionalidades, cada elemento puede estar anidado dentro de un paquete dentro del diagrama. Estos diagramas se usan comúnmente con la finalidad de proporcionar una organización visual de la arquitectura del software, los diagramas de paquetes pueden ser actualizados de manera sencilla a medida que el proyecto avanza y evoluciona.

En este tipo de diagramas es simple de construir y comprender, permitiendo visualizar los paquetes esenciales de todo el sistema, dentro de la investigación científica un diagrama de paquetes resulta una herramienta útil donde agrupar los comportamientos, datos e interacciones del software que se planea desarrollar. La composición de un diagrama de paquetes es simple, sin embargo, su construcción está basada en ciertos símbolos que se describen a continuación.

El diagrama de paquetes se construye a partir de ciertos componentes básicos, la imagen de un paquete indica la agrupación de elementos comunes basados en datos comportamientos o interacciones.

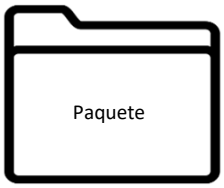


Fig. 6. 1 representación de paquete en UML

Las relaciones entre paquetes pueden emplearse de distintas maneras para representar diferentes dependencias entre los paquetes de un sistema, visualmente se representan como líneas discontinuas entre paquetes y se dividen en dos tipos, de acceso y de importación.

Acceso: En esta dependencia de paquetes indica que un paquete requiere la ayuda de las funciones de otro paquete, este tipo de relación esta estereotipada como <<access>>.

Fig. 6.2 Representación de la relación en UML

Importación: Indica que la funcionalidad se ha importado de un paquete a otro, esta relación se estereotipa como <<import>>.

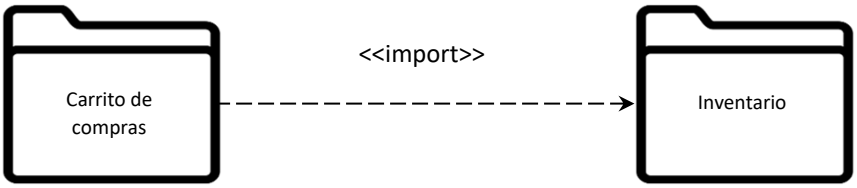


Fig. 6.3 Representación de importación en UML

Entradas y salidas en los paquetes

Además de las dependencias de funciones entre los paquetes, intercambian datos consecuencia de algún proceso de software, es decir, hay datos que pueden moverse entre los paquetes o ser el resultado final de la ejecución de las funciones en el software.

El software o una parte de este recibe una entrada, internamente realiza una serie de procedimientos con base en los requerimientos acordados previamente, para finalizar se entrega un resultado que puede ser el final o la entrada para el siguiente paquete de software de tal forma que el proceso pueda continuar. Esto puede verse como entradas entre varios paquetes o ingresados directamente por el usuario, así como salidas necesarias para continuar con algún proceso o información que debe entregarse al usuario.

En el diagrama de paquetes se pueden añadir las entradas y salidas de los módulos utilizando notas, dentro del diseño de diagramas de paquetes en UML no es imprescindible escribir cada entrada y salida del paquete ayuda al programador a saber cuál es el resultado esperado después del procedimiento y qué entradas corresponden al mismo.

Las entradas se pueden indicar por medio de un símbolo de nota con una esquina doblada y se adjunta al elemento del diagrama conectándolo mediante una línea punteada con el módulo donde servirá como entrada, en el caso de una salida se conecta desde el módulo hacia el símbolo de nota, donde se indicará la salida.

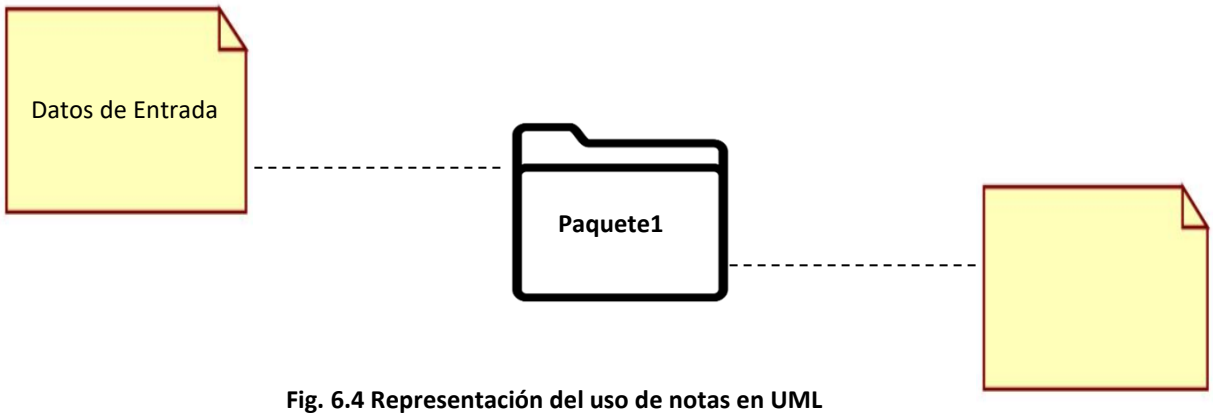


Fig. 6.4 Representación del uso de notas en UML

Cuando se utilizan notas en UML se realiza una unión por medio de una línea punteada hacia el paquete al que pertenece, si bien no es un símbolo exclusivo de entradas y salidas, puede ser útil añadir notas de las características que los datos deben tener al momento de entrar y salir de un paquete dentro del diagrama, esto puede ayudar a entender de mejor manera las funciones que ocurren dentro de un paquete cuya nomenclatura no sea tan descriptiva. A partir de la construcción de un diagrama de paquetes se pueden observar las secciones principales del software, así como las entradas y salidas correspondientes.

Diagramador UML

En esta sección se describe la construcción de un diagrama de paquetes utilizando la herramienta visual-paradigm en su versión online.

https://online.visual-paradigm.com/drive/#diagramlist:proj=0&new=PackageDiagram

Para la creación de diagramas en visual-paradigm es necesario generar un diagrama en blanco presionando la sección "*Start from blank*".

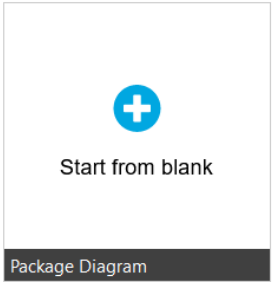


Fig. 6.5 Botón para crear un nuevo diagrama

Una vez que el espacio de trabajo se presenta en pantalla, se pueden añadir tantos símbolos de paquetes como sea necesario, esto se logra a través del panel del lado izquierdo insertando el símbolo de carpeta que indica un paquete, es posible modificar el tamaño y el nombre de cada paquete dando doble clic sobre el nombre predeterminado "*Package*" y escribiendo el deseado.

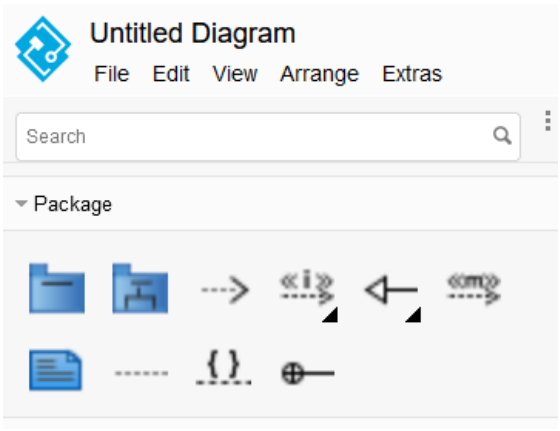


Fig. 6.6 Panel de símbolos

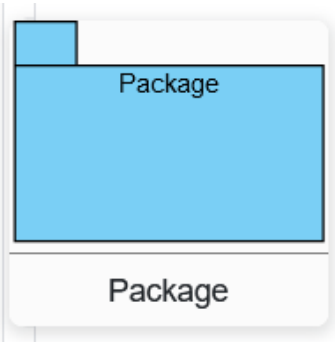


Fig. 6.7 Símbolo de paquete

El proceso de añadir y renombrar paquetes se repetirá hasta que se tenga el número de paquetes deseado, como se muestra en el ejemplo siguiente.

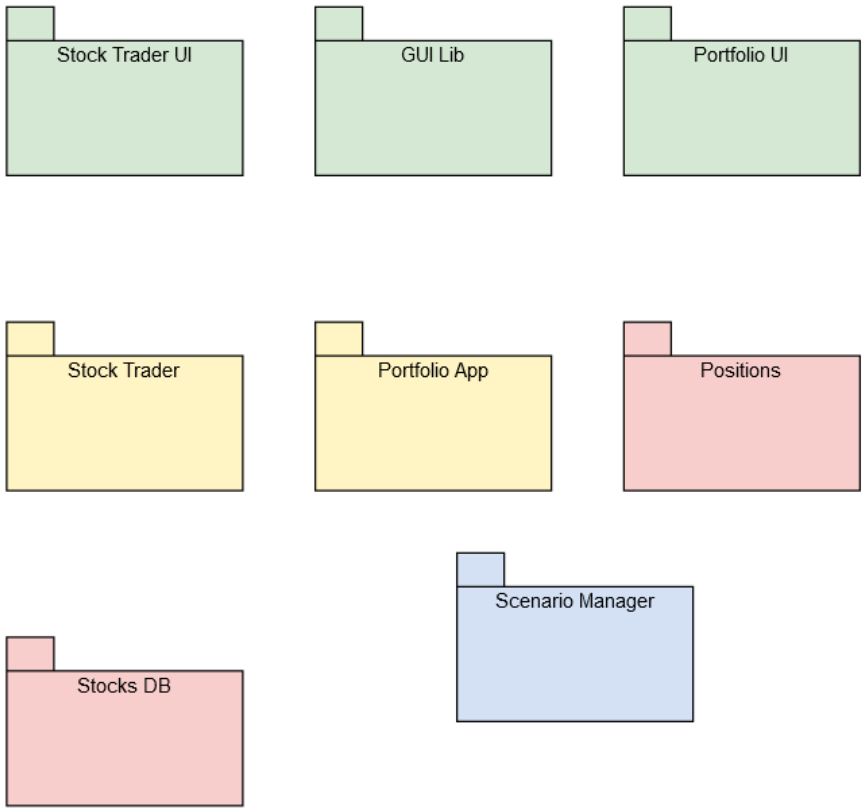


Fig. 6.8 Conjunto de paquetes que forman un sistema

Cuando se añaden los paquetes al área de trabajo, es momento de definir como están relacionados, para esto es necesario seleccionar el símbolo de la flecha punteada que se encuentra en el panel izquierdo con el cual se indican las uniones entre paquetes y el flujo de información entre estos.

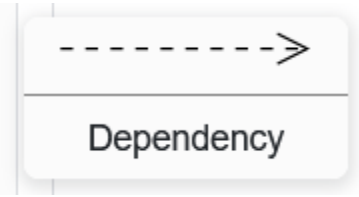


Fig. 6.9 Línea para indicar dependencia en UML

Al final el diagrama de paquetes con las relaciones ya establecidas puede verse como en la figura 6.10.

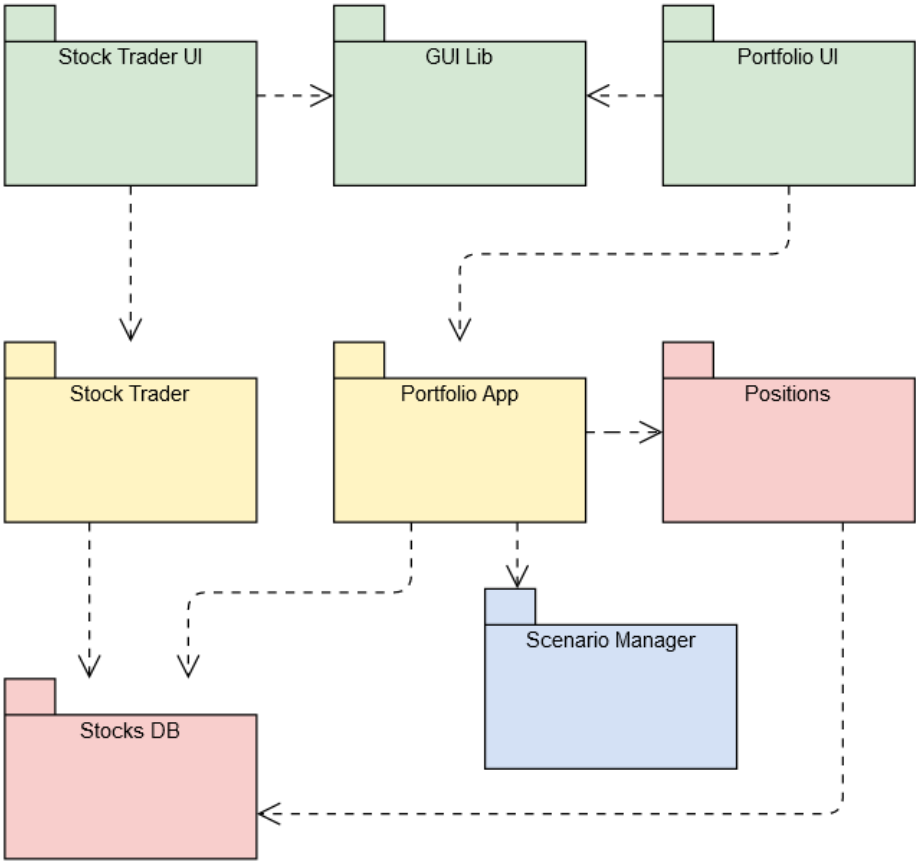


Fig. 6.10 Diagrama de paquetes

De esta forma podemos construir un diagrama de paquetes de forma sencilla con esta u otra herramienta semejante.

6.1.2 Directrices para el modelado del sistema

El diseño o modelado tiene como objetivo identificar y agrupar la estructura del software, así como sus características y conceptos que permitirán el desarrollo de un producto de calidad y establece ciertas reglas para el sistema de software que se desea crear.

Directrices para el diagrama de paquetes

Contemplar un diseño de la estructura y funcionamiento del software antes de escribir código resulta una herramienta útil para determinar las partes que deben crearse y cómo estas interactúan con el resto de las funcionalidades, además, es útil para la evolución del software. A continuación, se enumeran unos pasos para construir el diagrama de paquetes:

1. Identificar los paquetes o funcionalidades esenciales que conforman el sistema completo, estos paquetes contienen varios requerimientos del mismo tipo o que están muy relacionados entre sí.

2. Utilizando un software diagramador de UML como visual-paradigm se deben añadir los paquetes que se definieron en el punto anterior y renombrar cada paquete de tal forma que sea sencillo identificar unos de otros y su contenido.

3. Una vez que se han añadido todos los paquetes es momento de añadir las relaciones entre paquetes, no es necesario que cada paquete tenga una unión con el resto, pero es importante determinar que uniones son necesarias para que el flujo del programa se visualice de forma sencilla en el diagrama. Las uniones son representadas por una flecha punteada, que inicia en un paquete de donde sale la información o termina un proceso y apunta hacia donde debe continuar el flujo del programa.

4. Si es necesario se pueden añadir notas para aclarar las características de las entradas y salidas o aclarar el funcionamiento del paquete.

5. Discutir con los interesados el diseño de paquetes para obtener su aceptación de la estructura general del sistema.

6.2 Construcción

Durante el desarrollo de un sistema de software, el trabajo de construcción se enfoca en generar el código del software siguiendo los estándares y normas según el lenguaje y las tecnologías a usar. Los programadores son el rol que mayor participación tiene en esta fase del ciclo de vida del software.

6.2.1 Definir ambiente de construcción

La toma de decisiones sobre el ambiente de desarrollo es determinada por dos factores importantes, en dónde se ejecutará el software y el conocimiento de los desarrolladores sobre determinado lenguaje de programación.

- Algunos softwares son pensados para ser desarrollados como páginas web por lo que se requiere una infraestructura diferente para permitir que esta página esté disponible, contrario a un software que se ejecuta de forma local en algún equipo de cómputo personal, esta decisión permite al equipo discernir entre un conjunto de lenguajes de programación para seleccionar el más adecuado para desarrollar.

- Otro punto que puede ayudar a determinar las tecnologías de desarrollo son los procesos que ejecuta el software, si este software requiere analizar rápidamente grandes cantidades de información entonces los involucrados podrían elegir un lenguaje o plataforma de desarrollo que se encuentre entre los más rápidos del mercado, por el contrario si el software requiere una mayor potencia en los gráficos e interfaces muy llamativas aunque no sea rápido entonces debe seleccionarse una tecnología que cumpla con estos requerimientos.

- El conocimiento de los desarrolladores sobre alguna tecnología suele ser un factor importante para la toma de esta decisión pues si es un lenguaje muy reciente, los programadores deberán tomar algún tiempo extra en conocer cómo construir software con una nueva tecnología y esto podría retrasar el arranque del desarrollo.

- Se recomienda hacer una lista de las tecnologías de desarrollo más utilizadas por la comunidad y a partir de ahí tomar en cuenta características vitales del software y conocimiento de los programadores sobre estas plataformas para finalmente tomar la decisión.

6.2.2 Conceptos de Frameworks

A continuación, se describen dos framework utilizados comúnmente para el desarrollo de software.

Spring Framework

Es un marco de trabajo del lenguaje Java proporciona un modelo de programación y permite configurar aplicaciones empresariales, tiene soporte de infraestructura a nivel aplicación, se enfoca en la parte “interna” de las aplicaciones está basado en la arquitectura Modelo-Vista-Controlador, durante el desarrollo se pueden crear módulos acoplados libremente, que no dependen de las librerías en código.

Dentro del desarrollo solamente se toman en cuenta los paquetes y clases necesarios para el proyecto, el resto de ellos pueden ignorarse. El proceso de realizar pruebas dentro de Spring es simple, porque el código dependiente del entorno se mueve hacia el framework, con el uso de JavaBeanstyle se utiliza la inyección de dependencia para hacer pruebas para esto se debe hacer uso de datos dummies y observar el resultado.

Otra característica destacable de Spring es el acceso a datos por medio de una API llamada Java Database Connectivity (JDBC) también soporta tecnologías de acceso a datos avanzadas como el uso de DAO, XML y ORM como Hibernate, y se integra de manera sencilla con bases de datos NoSQL como MongoDB.

https://spring.io/projects/spring-framework#learn

Django Framework

Es conocido como el mayor framework web basado en Python cuenta con una interfaz de administración potente, formularios model-vased, utiliza ORM (Objetc Relational Mapper) asignando objetos a tablas de una base de datos. Permite construir aplicaciones en poco tiempo ya que hay muchas funciones ya implementadas, bastara con adaptarlas.

Cuenta con una gran versatilidad, ya que funciona con cualquier framework del lado del cliente, puede "responder" en casi cualquier formato tales como HTML, JSON, XML, de forma interna también permite utilizar diferentes motores de bases de datos y ampliar componentes.

Al estar escrito en Python permite portabilidad en las aplicaciones ya que se ejecuta en muchas plataformas tales como Linux, Windows, Mac OS X. El código de Django está basado en patrones de diseño que permite la creación de código mantenible y reutilizable.

En una investigación científica se requiere ver los resultados y avances de forma evolutiva, al igual que cambian los requerimientos en el software, cambia la manera en que el proceso de desarrollo se lleva a cabo, los modelos evolutivos son iterativos, permiten desarrollar versiones cada vez más complejas del software.

6.2.3 Manejo de Versiones

Al escribir software es común organizar el desarrollo en una estructura de carpetas que representan una sección o funcionalidad esto permite a los diferentes programadores trabajar en funciones separadas y cada uno puede realizar cambios en secciones diferentes del árbol de carpetas, el control de versiones ayuda a resolver el problema del seguimiento de cambios individuales de cada desarrollador, resolviendo incompatibilidades cuando varios programadores modifican las mismas secciones de código.

El manejo de versiones o control de código fuente es una práctica que permite rastrear, gestionar y verificar los cambios del software a lo largo del desarrollo, estos sistemas de control de versiones ayudan a que el equipo de desarrollo pueda trabajar de forma rápida e independiente. Este software realiza un seguimiento de los cambios que cada programador realiza, cuando ocurre un error o es necesario "retroceder" en alguna modificación reciente se pueden revisar versiones anteriores en donde el software iba por el camino adecuado.

Utilizar un sistema de versionamiento permite resguardar el código fuente que ha sido escrito hasta el momento, las fallas de hardware son un riesgo latente dentro del desarrollo de software por lo que gestionar las versiones a través de un sistema permite que siempre exista en la nube una versión funcional del software.

6.2.4 Configuración del ambiente de implementación elegido

Spring

Para este framework es necesario tener instaladas las siguientes herramientas

- ✓ JDK que corresponda al sistema operativo que está usando
- ✓ IDE Eclipse JEE.

En la página oficial de apache Maven [https://maven.apache.org/download.cgi] en el apartado *files* se encuentran los enlaces de descarga, seleccionar el archivo que se encuentra al final de la lista comprimido en ZIP.

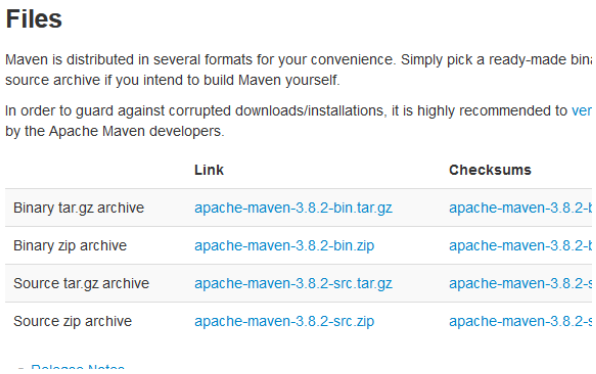


Fig. 6.11 Archivos disponibles para descarga

Al descargar el archivo es necesario descomprimirlo, para obtener la carpeta apache-maven-3.8.2, copiar esta carpeta dentro del disco C.

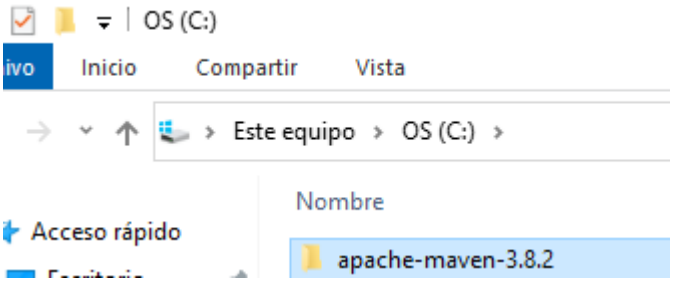


Fig. 6.12 Directorio de la carpeta apache-maven

A continuación, se deben añadir las variables de entorno a Maven, en la barra de búsqueda escribir "variables de entorno" y seleccionar la opción Editar las variables de entorno del sistema.

Se abrirá una ventana llamada propiedades del sistema, presionar el botón "variables de entorno"

En la ventana llamada Variables de entorno, seleccionar el botón "Nueva".

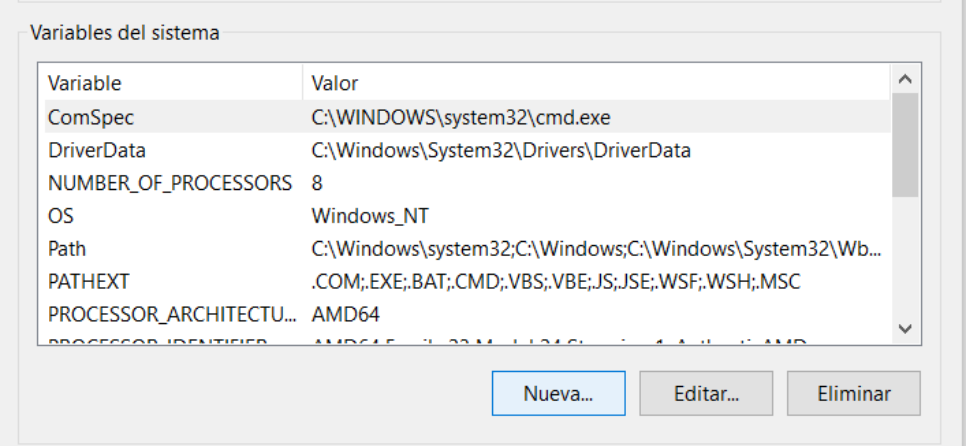


Fig. 6.13 Variables de entorno

Llenar el formulario que se despliega de la siguiente manera, al terminar de llenar los campos dar clic en Aceptar.

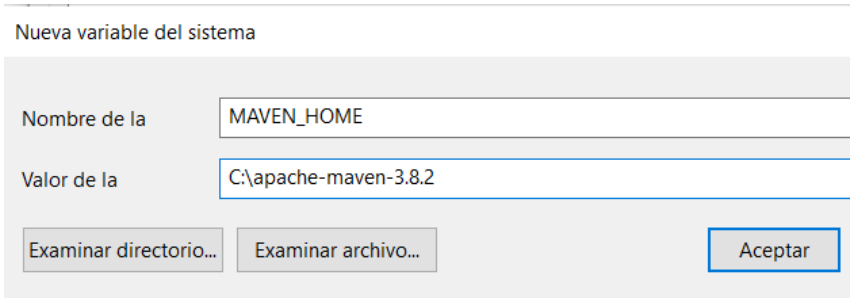


Fig. 6.14 Añadir variable de entorno

A continuación, dar doble clic a la variable llamada Path, se despliega una nueva ventana llamada Editar variable de entorno, aquí se debe añadir la ruta donde se encuentra la carpeta que descargamos, en este caso la ruta es C:\apache-maven-3.8.2, al finalizar dar clic en Aceptar.

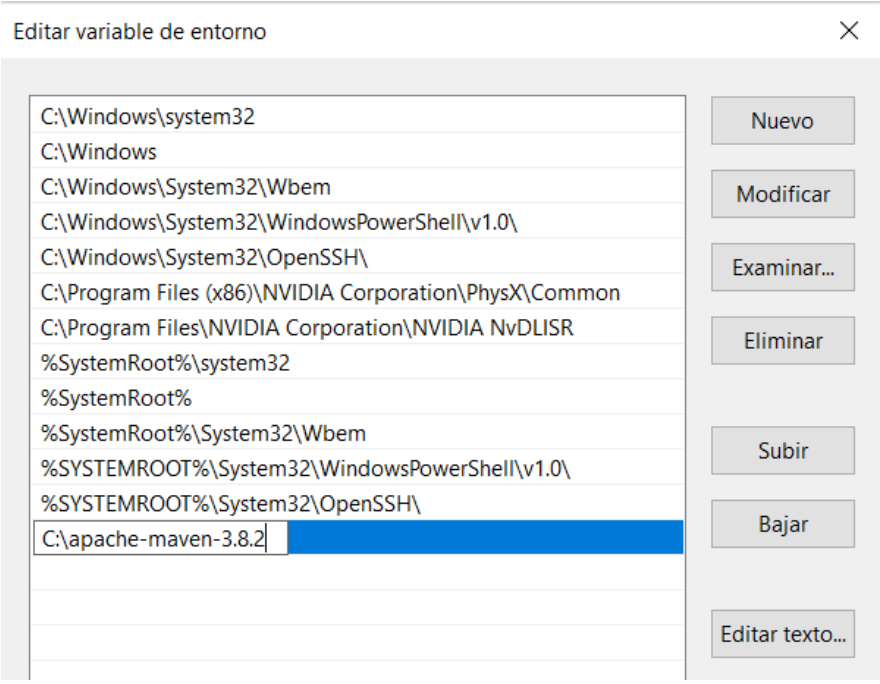


Fig. 6.15 Añadir directorio al path del sistema

Para comprobar que la instalación fue exitosa abrir una consola de comandos y ejecute el comando mvn -v, si la instalación fue exitosa muestra la versión instalada.

Fig. 6.16 Verificar instalación desde consola

Django

Es un framework de desarrollo web basado en Python, Django en su versión 3.2.7 requiere la instalación de Python 3.9.7.

Para instalar Python en el sistema operativo Windows acceder a la página oficial [https://www.python.org/downloads/release/python-397/] en la sección de descargas

Dar clic en el botón “Download Python 3.9.7”

Fig. 6.17 Botón para descargar Python

Una vez que termine la descarga dar doble clic en el archivo Python-3.9.7-amd64

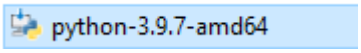


Fig. 6.18 Archivo descargado

Dentro del instalador verificar que las casillas en la parte inferior se encuentren marcadas, posteriormente dar clic en Install Now

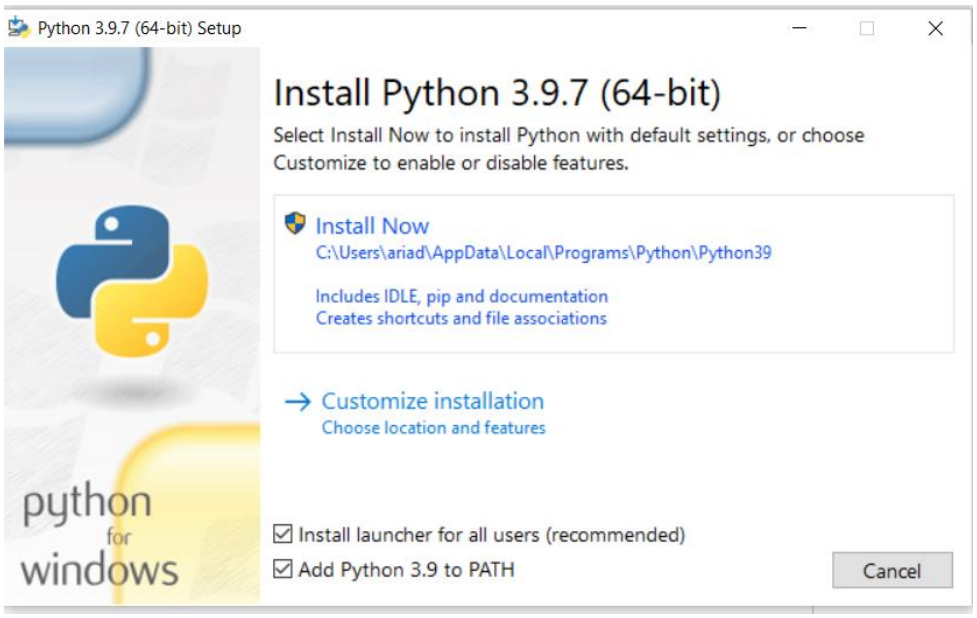


Fig. 6.19 Instalador de Python

Al finalizar la instalación se muestra la siguiente ventana indicando que la instalación fue exitosa

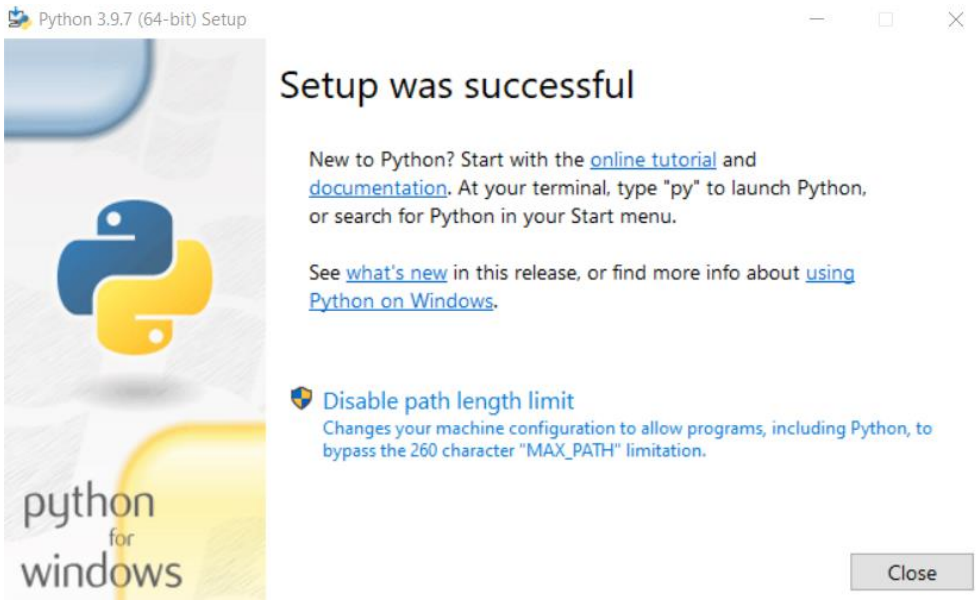


Fig. 6.20 Instalación finalizada

Posterior a la instalación de Python deberá instalar Django es necesario ejecutar el comando pip install django==3.2.7

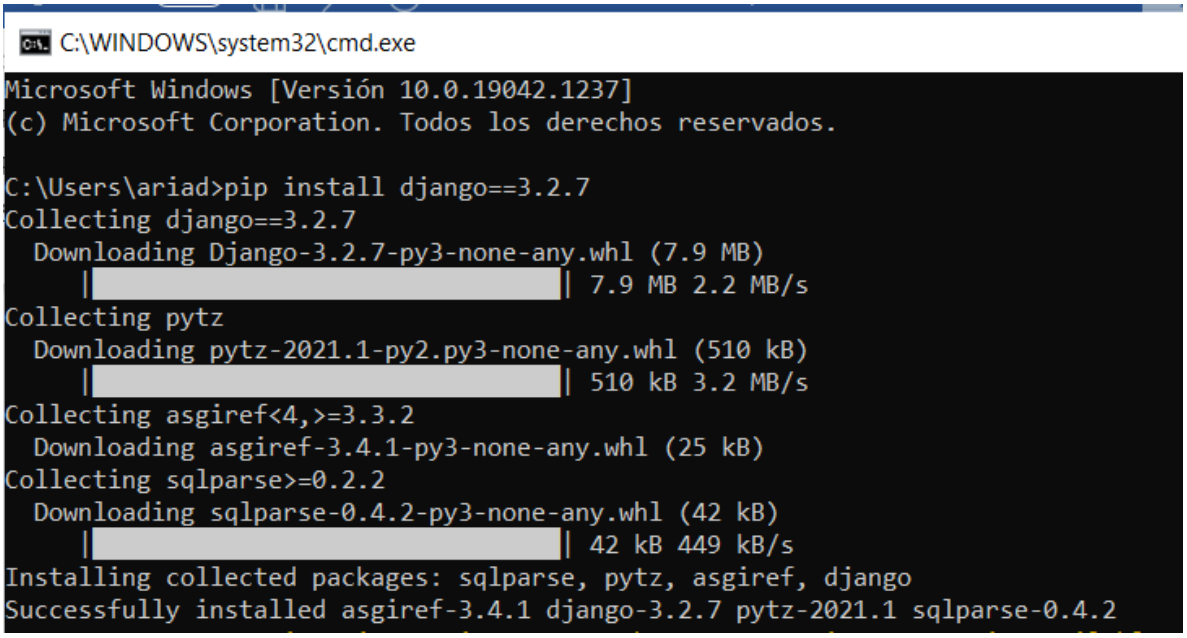


Fig. 6.21 Instalación de Django desde consola

Como parte de la configuración del framework se recomienda crear una carpeta en donde almacenar los proyectos, es necesario copiar la ruta de esta carpeta pues será de utilidad más adelante en la configuración.

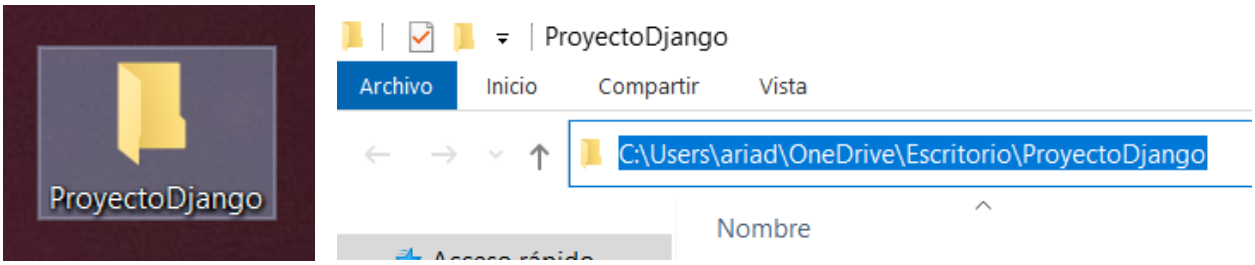


Fig. 6.22 Ubicación de la carpeta Proyecto Django

Utilizar una consola de sistema y acceder al directorio de la carpeta ProyectoDjango

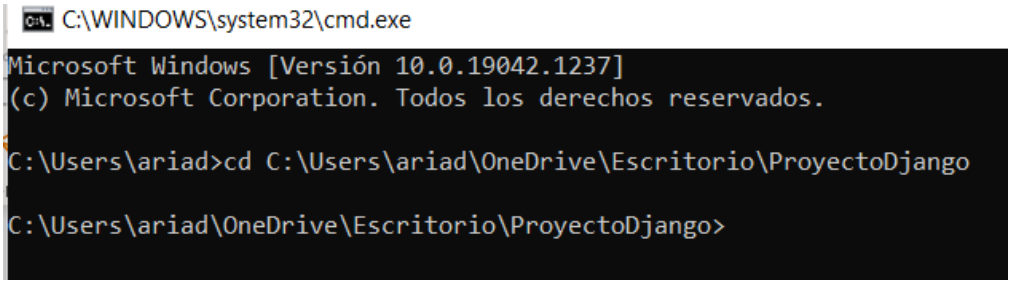


Fig. 6.23 Acceso a la carpeta desde consola

Crear un nuevo proyecto con el comando
django-admin startproject “nombredelproyecto”

Fig. 6.24 Crear un nuevo proyecto de Django

Dentro de la carpeta seleccionada para guardar el proyecto se genera una nueva con el nombre asignado al proyecto, a partir de este momento ya puede comenzar a trabajar con Django.

6.2.5 Manejador de versiones Github

Es una plataforma para la gestión de versiones dentro del desarrollo de software, aloja proyectos y permite el desarrollo colaborativo, esta plataforma cuenta con una versión gratuita que funciona como repositorio público de código y brinda herramientas para el versionamiento.

Ofrece herramientas para el trabajo en equipo tales como sistema de seguimiento de problemas que permite informar al resto de involucrados que inconvenientes han surgido, un conjunto de comandos para la gestión de versiones y acceso al código, su organización en forma de árbol permite visualizar de manera sencilla en que trabaja cada programador, así como las actualizaciones de cada sección del sistema.

Github (Github, s.f.) utiliza un sistema de ramas características en donde cada desarrollador puede dividir la rama principal para generar un repositorio local aislado del resto en donde puede realizar los cambios que desee sin afectar a la rama maestra que es donde se encuentra el código original del proyecto, una vez que se terminen de escribir los cambios en alguna rama y se considera que pueden fusionarse a la rama principal se unen por medio de la ejecución de algunos comandos y de esta forma los cambios se vuelven efectivos.

Para comenzar a trabajar con GitHub es necesario crear una cuenta en la página oficial de GitHub. Una vez que se tiene acceso a la cuenta para crear un nuevo repositorio basta con ir al botón en color verde llamado *New* ubicado en el panel izquierdo de la página web.

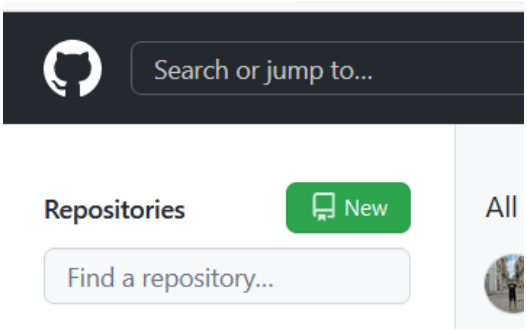


Fig. 6.25 Botón para crear un nuevo repositorio

Se desplegará un sitio en donde se podrán definir ciertas características del nuevo proyecto, en esta se muestra el nombre del propietario y el nombre que se desea asignar al repositorio, se puede añadir una descripción, así como la visibilidad que puede ser pública o privada. En la sección final de esta página se debe indicar como inicializar el nuevo repositorio, generalmente se añade un archivo llamado *Readme* en donde se puede añadir una descripción más detallada del proyecto.

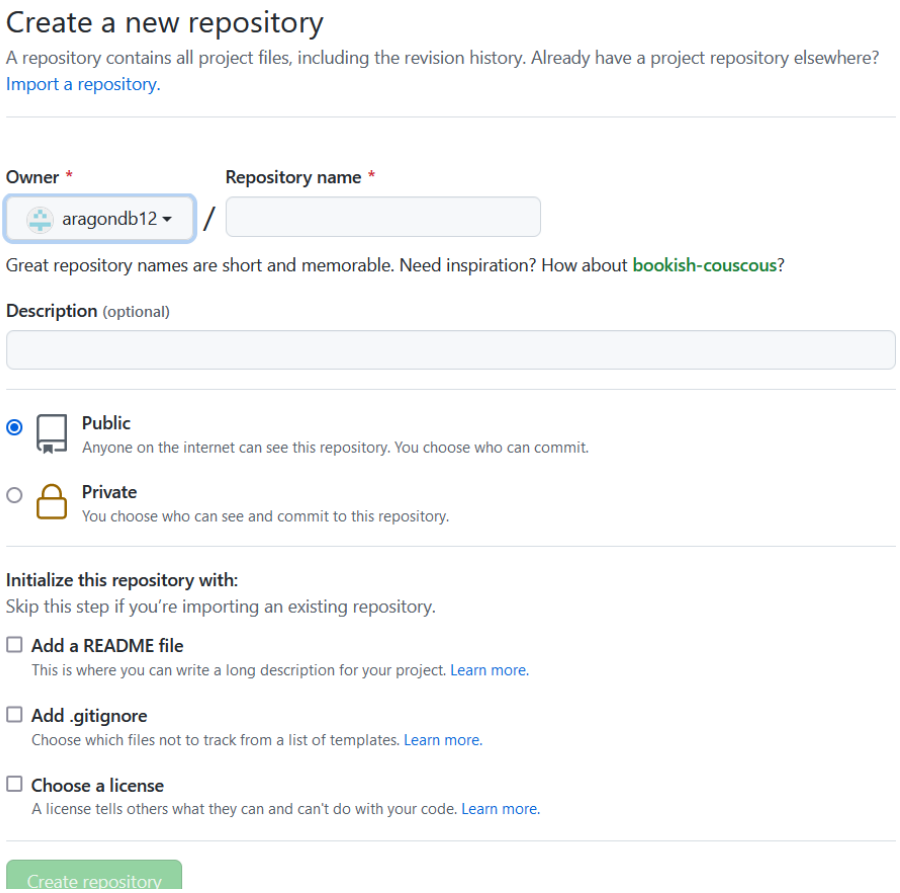


Fig. 6.26 Configuración de repositorio

Al finalizar el proceso se muestra la vista siguiente del repositorio nuevo que se ha creado.

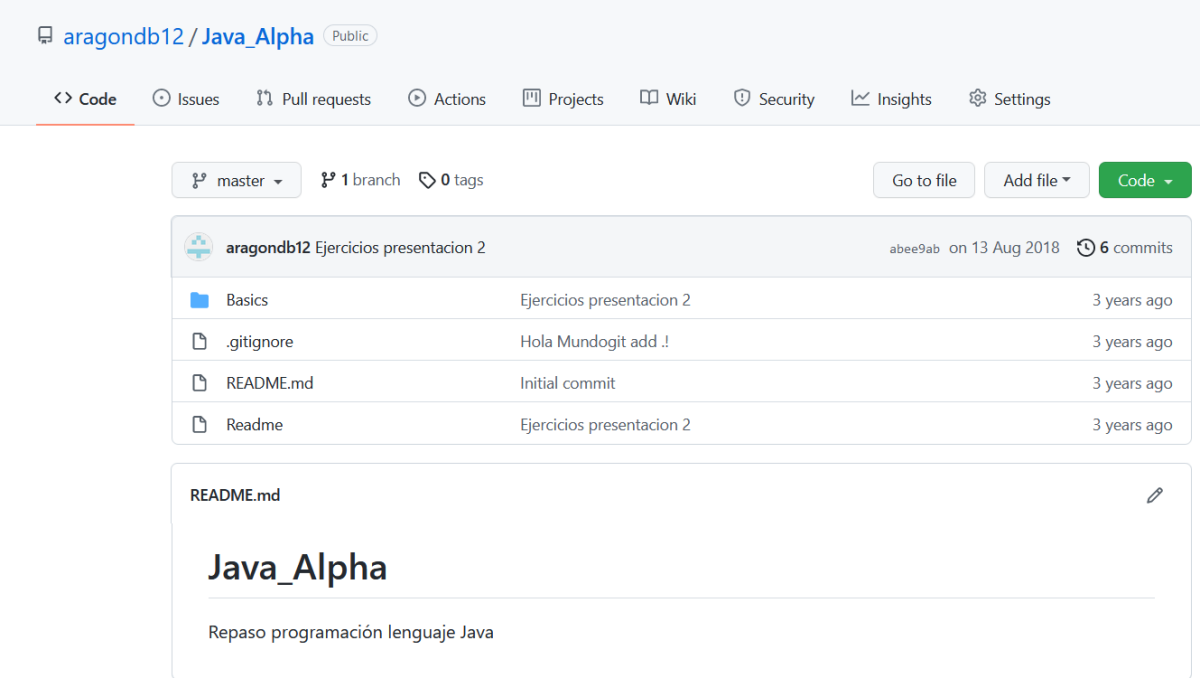


Fig. 6.27 Vista de repositorio nuevo

La creación de ramas es uno de los procesos más importantes que github nos permite usar para permitir el trabajo conjunto, cada rama genera diferentes versiones de un repositorio, cuando se realizan cambios en alguna rama de trabajo el desarrollador puede ver cómo estos cambios afectaran al proyecto maestro o rama principal cuando se integren.

Para generar una rama nueva basta con pulsar el botón master e introducir el nombre de la nueva rama, posteriormente hacer clic en Create Branch: “nombre” from ‘master’.

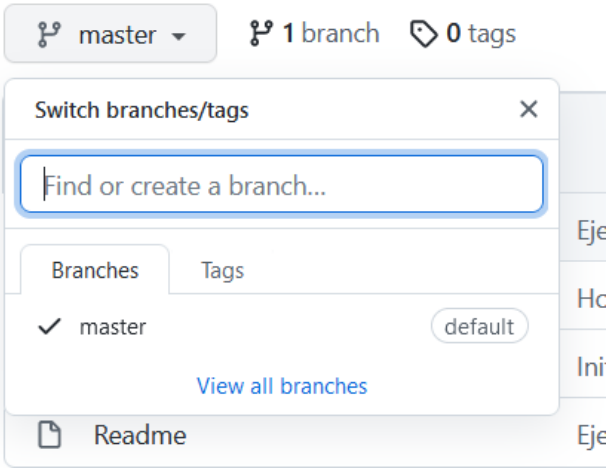


Fig. 6.28 Menú para crear nueva rama

A continuación, se detallan las funcionalidades más importantes de GitHub (Github, s.f.)

- git init [nombre del proyecto]: crea un nuevo directorio en donde ubicara un repositorio para el proyecto de manera local.
- git clone nombredeusuario@host:/path/to/repository: se usa para copiar un repositorio existente que se encuentra almacenado en un servidor remoto
- git add: agrega archivos al área de preparación, el comando "git add <temp.txt>" añade el archivo temp.txt
- git commit: crea un nuevo registro de los cambios en el repositorio
- git status: muestra la lista de los archivos que han sido modificados junto con los que están por ser preparados o confirmados
- git push origin master: envía confirmaciones locales a la rama maestra del repositorio remoto.

6.2.6 Manejador de Bases de Datos

Tener un adecuado manejo de datos cuando se desarrolla Software Científico es importante, pues procesar estos datos permite validar o rechazar la hipótesis planteada en la investigación científica, obtener información, analizar resultados o tomar decisiones respecto al proyecto de investigación.

Por lo que se recomienda utilizar un manejador de base de datos que permita almacenar permanentemente los datos y resultados del procesamiento hecho dentro del sistema de software.

PostgreSQL es uno de los sistemas de gestión de bases de datos relacionales de código abierto más utilizados actualmente, permite definir funciones personalizadas dependiendo del lenguaje que se utiliza, para el lenguaje de programación Python tiene PL/Python y para el caso de java existe PH/Java.

Para instalar PostgreSQL acceder al enlace de descargas [https://www.enterprisedb.com/downloads/postgres-postgresql-downloads], descargar la versión más reciente, dando clic en el botón download correspondiente al sistema operativo que este instalado en el equipo.

Fig. 6.29 Descarga de PostgreSQL

Al terminar de descargar el instalador es necesario ejecutar el mismo, posteriormente muestra una serie de ventanas, en la primera de ellas se muestra un mensaje de bienvenida basta con dar clic en el botón siguiente para avanzar en la instalación.

En la siguiente ventana se puede especificar la carpeta en donde se instalarán los archivos del programa, para continuar presione siguiente.

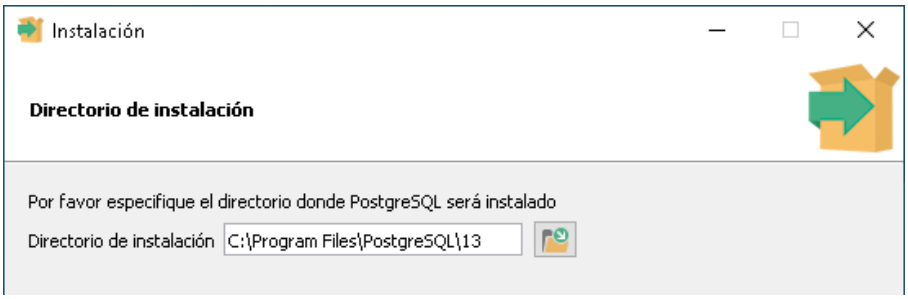


Fig. 6.30 Ruta de instalación

En la ventana componente deben estar seleccionadas todas las casillas y continuar a la siguiente ventana, en donde nos muestra la ruta de la ventana anterior añadiendo una carpeta llamada /data, aquí se almacenan los datos, esta ruta puede ser modificada o dejada por defecto.

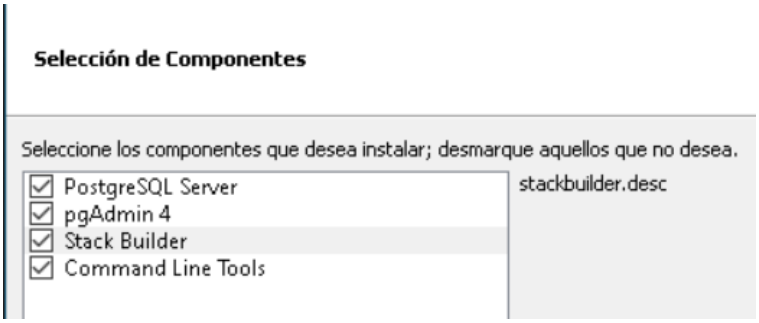


Fig. 6.31 Selección de componentes

Los manejadores de bases de datos permiten añadir contraseñas y usuarios para poder acceder, por lo que el siguiente paso en la instalación será definir una contraseña para el administrador, es decir el usuario principal del manejador y quien puede modificar como y donde se almacenan los datos.

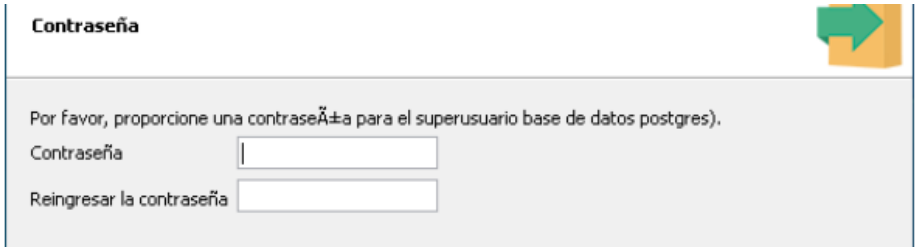


Fig. 6.32 Asignación de contraseña

Ahora es momento de asignar un puerto a través del cual PostgreSQL escucha las conexiones, se encuentra por defecto el 5432, otros manejadores utilizan el puerto 3306 pero en este caso dejaremos el que se sugiere.

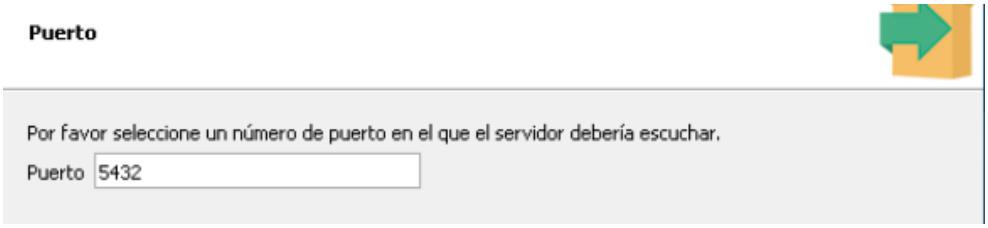


Fig. 6.33 Selección de puerto

Lo siguiente es indicar la configuración regional, es importante para que los caracteres especiales puedan ser reconocidos en la escritura y lectura de datos.

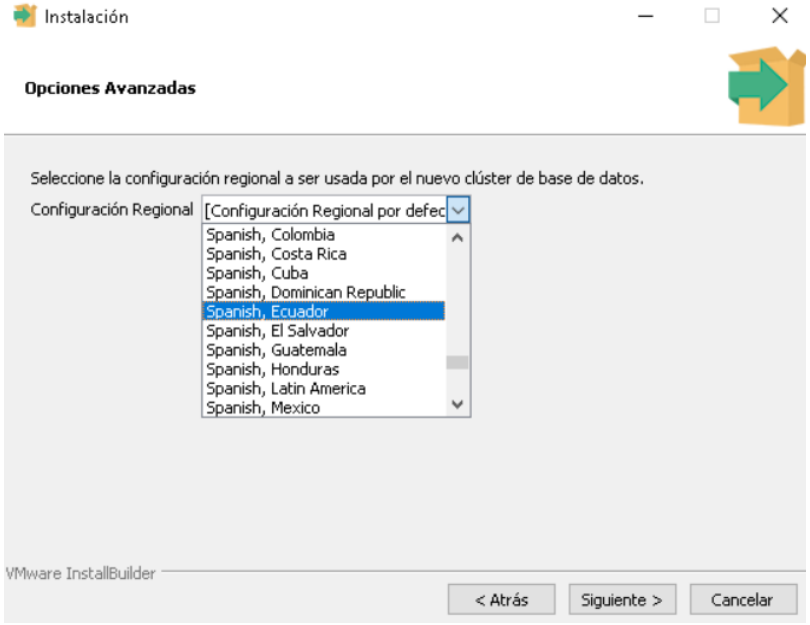


Fig. 6.34 Configuración regional

A continuación, se muestra un resumen de la configuración seleccionada, pasar a la siguiente ventana.

Fig. 6.35 Resumen de configuración

La configuración está terminada y puede iniciar con la instalación dando clic en el botón siguiente.

Fig. 6.36 Inicio de instalación

El proceso de instalación se ejecuta mostrando una barra de avance

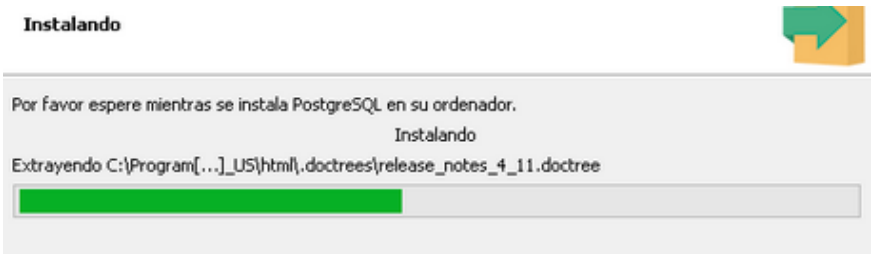


Fig. 6.37 Progreso de instalación

Cuando todos los componentes hayan sido instalados dar clic en el botón Terminar

Fig. 6.38 Instalación finalizada

Verificación de la instalación

Para verificar que PostgreSQL se instaló de manera correcta, en la barra de búsqueda escribir la palabra Servicios, abrir la aplicación.

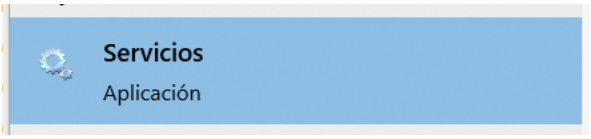


Fig. 6.39 Aplicación de servicios

En la ventana de Servicios ir a la sección servicios locales y buscar el servicio llamado postgresql.

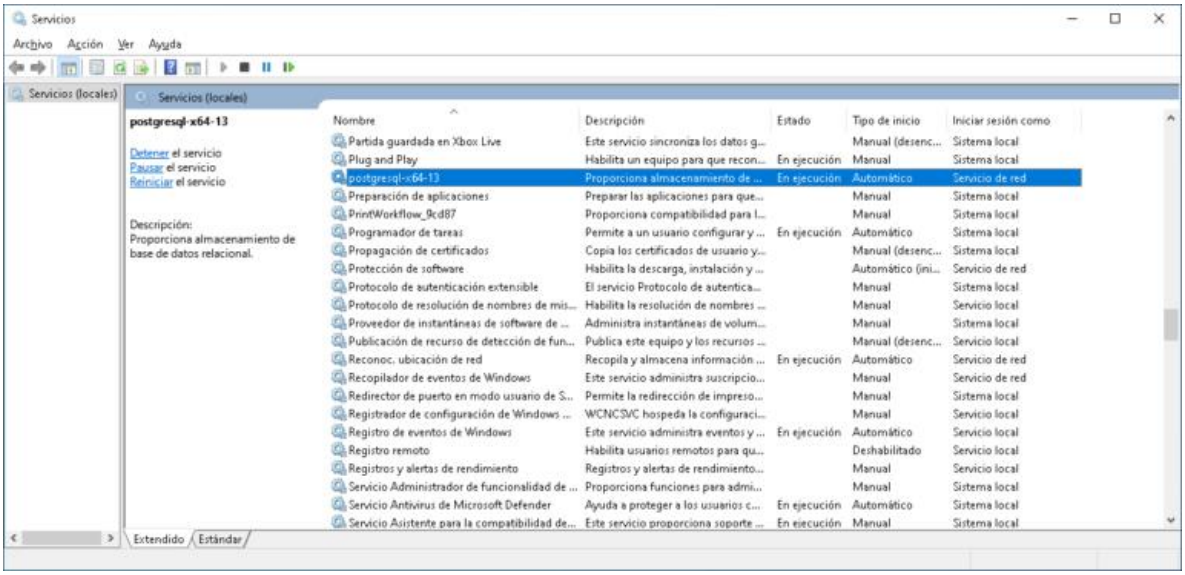


Fig. 6.40 Ventana de servicios

Ahora podemos ingresar al programa llamado pgAdmin que instalo junto con la base de datos, al ejecutar pgAdmin.

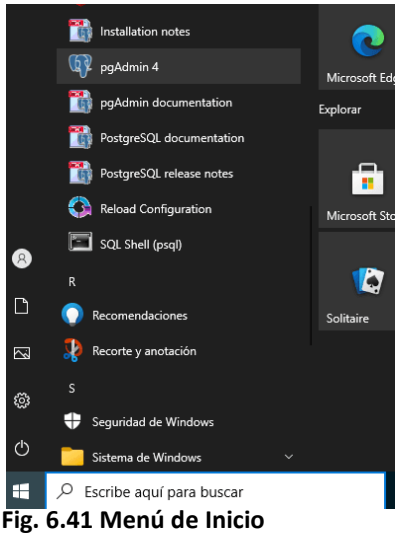


Fig. 6.41 Menú de Inicio

Fig. 6.42 Ejecución de pgAdmin

Al iniciar el programa se requiere ingresar la contraseña que asignamos durante la instalación.

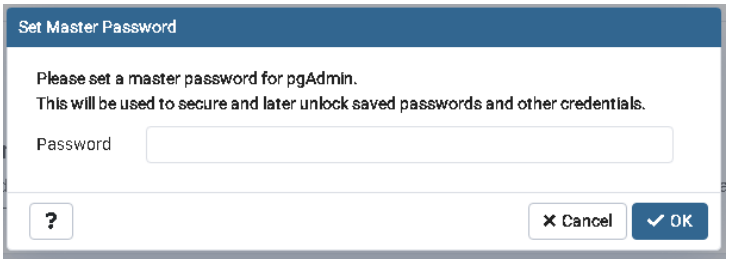


Fig. 6.43 Ingresar contraseña

En el panel izquierdo abrir el grupo de Servers y después el grupo de databases, dentro de este hay una base de datos llamada postgres.

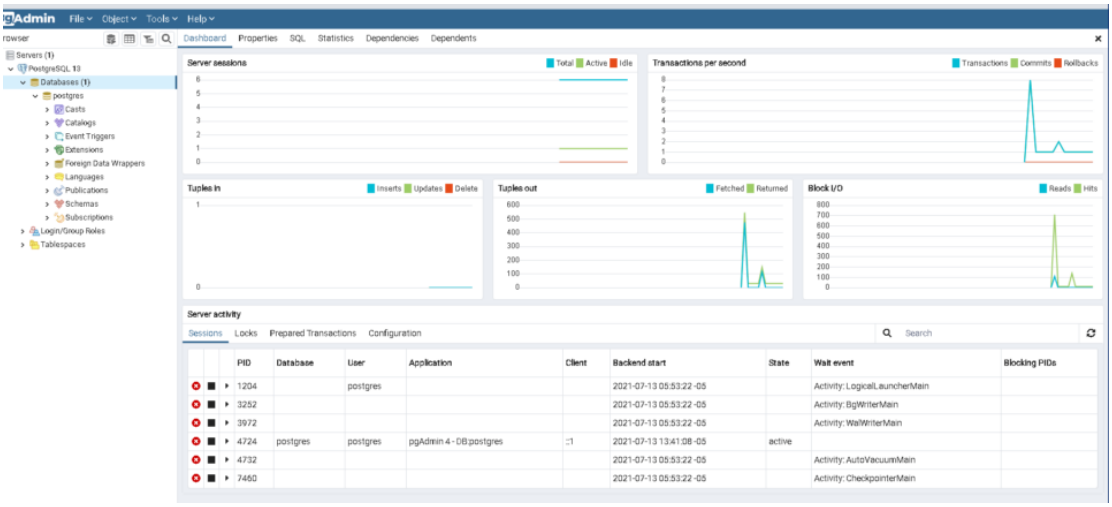


Fig. 6.44 Ventana de postgreSQL

Aquí se encuentra la base de datos instalada y activa para poder ser utilizada para el sistema que será desarrollado.

6.3 Definir ambiente de construcción

Una vez definidos los conceptos anteriores, se puede tomar la decisión del ambiente de construcción, que tiene como objetivo seleccionar las tecnologías que serán utilizadas por los programadores para desarrollar el software. Se trata de establecer las características como la plataforma sobre la que se ejecutará, el tipo de manejador de base de datos utilizará, manejador de versiones, etc. Con la finalidad de ayudar a los involucrados a definir el ambiente de construcción se proponen las siguientes directrices.

6.3.1 Directrices para definir el ambiente de desarrollo

La tabla Ambiente de Desarrollo permite a los involucrados definir y conocer las tecnologías que se utilizarán durante el desarrollo. Ayuda en la toma de decisiones sobre las tecnologías que serán utilizadas para el desarrollo, en ella se detalla que lenguajes de programación, Frameworks, manejadores de bases de datos etc. Los pasos para completarla son:

1. Determinar las características técnicas del software
    a) Aplicación web o de escritorio,
    b) Conexión con manejador de Base de Datos
    c) Uso de software externo
    d) Conocimiento del lenguaje por parte de los programadores

    Una vez se toma la decisión del lenguaje de programación, esta información debe añadirse en la fila llamada Lenguaje seleccionado en la tabla de Ambiente de Desarrollo

2. Una vez que se eligió el lenguaje de programación para el desarrollo es posible seleccionar un framework en la tabla de Ambiente de Desarrollo

3. La tecnología donde se almacenarán los datos que interactúan con el software debe ser seleccionada dependiendo de las necesidades del proyecto, es decir pueden ser relacionales o no relacionales.

4. Seleccionar el manejador de base de datos, tomando en cuenta la cantidad de programadores que estarán trabajando sobre el proyecto de manera simultánea.

6.3.2 Directrices para el control de versiones

El versionamiento permite gestionar los cambios y la evolución del software, así como la trazabilidad de los requerimientos y como han cambiado a través del avance del desarrollo.

1. Desde la página de github se debe generar un nuevo proyecto de forma pública para que el resto de involucrados puedan acceder a este.

2. Posteriormente será necesario instalar el software de github de forma local en cada equipo utilizado para desarrollar código.

3. A continuación, es necesario copiar el repositorio del proyecto, una buena práctica es crear ramas secundarias del proyecto y después copiar dichas ramas al repositorio local, esto se logra a través del comando git clone.

4. El desarrollo comienza al crear nuevos archivos de código y directorios dentro del repositorio local, por lo que cada vez que se decida comenzar a escribir una nueva funcionalidad del sistema se generan nuevos archivos. Cuando se considere que una funcionalidad esta completada se recomienda utilizar el comando git commit para generar un nuevo registro de cambios.

Finalmente se deberá utilizar el comando git push orign master cuando la funcionalidad ya ha sido probada y los involucrados están de acuerdo en añadirla a la rama maestra.

Como ayuda se propone construir una tabla como la siguiente:

Ambiente de Desarrollo			
Características técnicas	*Tecnología*	*Versión*	*Descripción*
Lenguaje seleccionado			
Framework			
Manejador de base de datos			
Manejador de versiones			

Fig. 6.45 Tabla Ambiente de Desarrollo

6.4 Pruebas

Durante la construcción del software pueden surgir errores o inconvenientes, estos pueden venir desde restricciones por la tecnología utilizada, un mal diseño de la solución, o errores en el código, por esto se deben realizar de forma constante pruebas a la solución, desde pruebas sobre un módulo del sistema y como un todo.

Las pruebas de software pertenecen también al alfa de Sistema de software y deben ser planeadas con anticipación, basándose en los requerimientos y el diseño acordados. El resultado de las pruebas determina la calidad del software y permiten descubrir defectos o fallas en el desarrollo y permiten al equipo tener un plan de solución ante tales problemas. La detección y corrección en etapas tempranas evitan que dichos fallos se acumulen al pasar a etapas siguientes, hasta que llegan al despliegue de la versión final del software.

En el desarrollo de una solución de software es necesario generar una estrategia de ejecución de pruebas con la finalidad de realizar las pruebas adecuadas, en las etapas iniciales las pruebas se enfocan en un solo componente (pruebas unitarias), conforme el proyecto avanza se revisa un grupo de componentes relacionados (pruebas de integración), buscando errores en los datos

de entrada o salida, así como en la lógica del procesamiento dentro de cada componente. Cuando el software integra varios módulos y se convierte en un sistema complejo, se deben ejecutar pruebas de orden superior (pruebas del sistema), en este punto es visible si existen diferencias entre los requerimientos establecidos previamente.

Pruebas unitarias

Estas pruebas consisten en verificar de forma individual funciones o métodos del sistema, dado que son revisiones especificas se pueden realizar de manera automatizada por medio de un servidor de integración continua (continuous integration). Idealmente cuando se escriben pruebas unitarias se deben modularizar de tal forma que no se pueda descomponer en más funciones o métodos, es decir hasta obtener pruebas para una unidad de código.

Las pruebas unitarias verifican que los tipos de parámetros sean los adecuados dentro de la función, así como el tipo de valor que es retornado como resultado, idealmente no deberán tener ningún tipo de dependencia externa, esto podría ser una base de datos o alguna interacción más allá de la unidad que está siendo probada.

Pruebas de caja blanca

En las pruebas de caja blanca se tiene acceso al código fuente por lo que se conocen detalles procedimentales del software, de esta forma es posible comprobar que el flujo de la ejecución sea el adecuado ya sea en pruebas a secciones del software o al sistema completo.

El uso de pruebas de caja blanca es útil para verificar el funcionamiento lógico del código y así detectar errores que ocurren debido al cambio de valores en variables dentro del código, permiten probar cada una de las condiciones existentes en el programa, identificar entradas, salidas y estudiar las relaciones entre ellas. En estas pruebas se desarrolla un conjunto de valores para condiciones o variables del código que actúan como entrada al programa y se definen sus respectivas salidas esperadas posterior al procedimiento, a estas se les conoce como casos de prueba (Pressman, 2000).

Las características de las pruebas de caja blanca son:

- Utilizar una estructura de control del diseño procedimental
- Garantizar que se ejecutan por lo menos una vez todos los caminos independientes del software
- Verificar el funcionamiento de las decisiones lógicas en las estructuras de verdadero y falso
- Ejecutar todos los bucles en sus límites operacionales
- Ejecutar estructuras internas de datos para asegurar la validez

Las pruebas de caja blanca permiten asegurar que el código realiza el proceso especificado por los investigadores, dando fiabilidad al software y permite identificar errores durante la ejecución para su posterior corrección. Es importante definir algunos conceptos para la creación de casos de prueba y de esta forma diseñarlos de forma que los testers puedan apoyarse de estos para verificar el correcto funcionamiento del software aun cuando se den diferentes datos de entrada (Martínez, 2012).

Camino: “secuencia de todas las instrucciones de un programa de principio a fin” definido como la ruta de secuencias que se siguen en el código fuente, es decir, rastrea el procedimiento desde la entrada del software hasta que el programa arroja el resultado, siguiendo la estructura de código definida por los programadores.

Camino independiente: “Es cualquier camino del programa que incluye nuevas instrucciones de un proceso o una nueva condición”. Los caminos independientes son el conjunto de operaciones que se ejecutan fuera del flujo principal del programa, estos caminos se ejecutan cuando hay errores en los datos que se requieren en procedimientos específicos, por ejemplo, al ingresar un número cuando se esperaba una letra se generan excepciones o errores y el flujo principal del programa que se espera, no se ejecuta.

Pruebas de Integración

Su objetivo es verificar que un conjunto de módulos o funciones del sistema de software cuyo funcionamiento está relacionado entre ellos funcionan bien juntos. Generalmente estas pruebas siguen a las pruebas unitarias ya que al asegurar que la unidad individual de software funciona de manera adecuada, los fallos encontrados serán debido a la integración de estos módulos.

En este tipo de pruebas se verifican las interacciones entre módulos y en cómo se envían los parámetros (por valor o por referencia) entre las unidades de software definido durante la etapa de diseño de software, Al construir las pruebas de integración se pueden tomar como referencia los “caminos” utilizados en las pruebas unitarias de esta forma se asegura que los valores tomen interacciones ya conocidas permitiendo seguir un flujo entre módulos.

Una parte importante de realizar estas pruebas es la interacción de las funciones de software con la interfaz mostrada al usuario, debe existir armonía entre el proceso, el resultado obtenido y como deberá ser presentado en una interfaz que los investigadores sean capaces de entender.

Pruebas de sistema

Tienen como objetivo realizar una constante revisión al sistema para comprobar que la integración de las nuevas funcionalidades no ha afectado el rendimiento global del aplicativo, revisando subsistemas, interfaces y como se comunican con el resto de los elementos.

Permiten probar el sistema como un conjunto y como se relaciona con otros sistemas para verificar que las especificaciones funcionales y técnicas se cumplen sin ningún problema, estas pruebas se ejecutan después de que cada componente se ha probado de forma individual.

Pruebas de regresión

El nuevo código resultado de una actualización o de añadir nuevas funcionalidades al sistema no deberá afectar el funcionamiento del software ya probado por lo que es recomendable automatizar estas pruebas para ahorrar tiempo y esfuerzo, cuando se ejecutan estas pruebas se pueden encontrar errores en alguna de las siguientes clasificaciones.

Las pruebas de regresión pueden apoyarse de las pruebas unitarias ya existentes para verificar que los cambios en el sistema no afectan al software previamente validado, el documentar

pruebas anteriormente realizadas permite conocer el comportamiento del software por esto se recomienda guardar datos obtenidos durante la ejecución de pruebas.

6.4.1 Herramienta para la gestión de prueba

Una herramienta útil cuando se busca tener un control sobre los cambios necesarios para corregir errores o bugs encontrados durante la etapa de pruebas es Trello, que permite dar seguimiento a las incidencias y ayuda en la gestión de éstas. La herramienta de gestión Trello es útil cuando se deben tomar en consideración los errores encontrados durante las pruebas, permite al equipo visualizar el avance de la solución y verificar que errores ya han sido corregidos de manera satisfactoria. Trello al ser un sitio web funciona en diferentes plataformas

Esta herramienta cuenta con una interfaz sencilla de utilizar similar a un tablero en donde se pueden añadir los resultados de las pruebas y el requerimiento necesario para solucionar el error encontrado, Trello permite añadir diversos estatus y clave de incidencia de tal forma que sea más sencillo identificar cada una.

El equipo tiene control de estas incidencias y cada programador involucrado puede tomar una o varias y trabajar sobre ellas, conforme sean solucionadas el estatus se modifica y actualiza para el resto del equipo con acceso a Trello.

Para poder acceder a este recurso en necesario generar una cuenta en el sitio oficial de Trello, una vez que se ha creado una cuenta se puede crear un tablero que permita a los involucrados y principalmente a los desarrolladores gestionar los cambios o problemas encontrados durante la etapa de pruebas.

Para crear un nuevo tablero de gestión seleccionar la sección tableros en el panel izquierdo de Trello, posteriormente dar clic en crear un tablero nuevo y asignarle un nombre que permita identificarlo de forma rápida.

Fig. 6.46 Panel Trello

Fig. 6.47 Espacio de trabajo

Automáticamente se carga una nueva ventana con el formato estándar de un tablero en donde se muestran tres secciones Lista de tareas, En proceso y Hecho. Estas listas pueden personalizarse y añadir más de acuerdo con las necesidades de los involucrados.

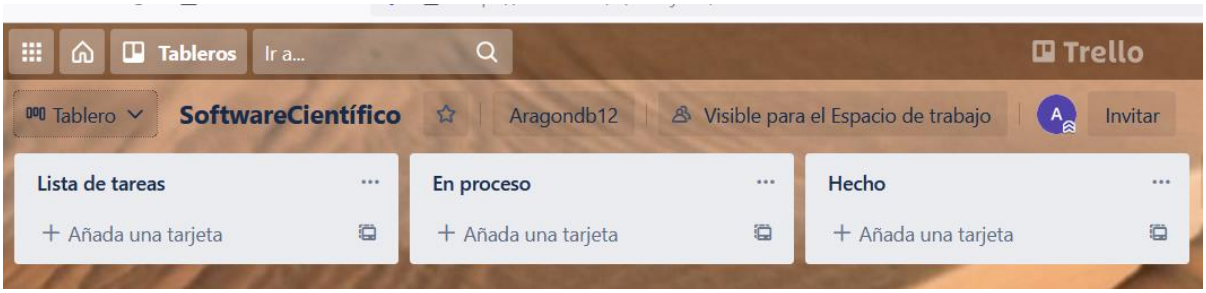


Fig. 6.48 Vista del nuevo tablero

Se deberán añadir tarjetas por cada cambio encontrado durante la etapa de pruebas, para añadir una nueva tarjeta basta con dar clic en la sección “Añada una tarjeta”, asignar el nombre de la prueba a la que corresponde este cambio y clic en “Añadir tarjeta”.

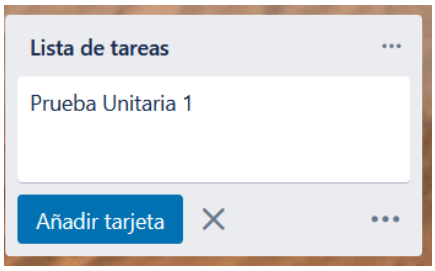


Fig. 6.49 Nueva tarjeta

Una vez creada la nueva tarjeta será necesario editar los detalles de esta, al dar clic sobre el nombre que se asignó a la tarjeta y se despliega el menú de detalles donde nos muestra un campo de descripción donde se añaden los detalles de la prueba que se ejecutó y cuál fue el resultado obtenido.

En el apartado actividad es posible añadir comentarios sobre el avance en la solución de algún problema o cambio, nuevas actividades, o informar al resto de programadores de los procedimientos realizados, esta sección funciona también como bitácora de quien ha realizado cambios o actualizaciones.

Fig. 6.50 Menú de definición de tarjetas

Es posible añadir a más involucrados en la ejecución de pruebas y el seguimiento de estas, esto se puede hacer danto clic en Miembros donde se despliega una lista de los miembros con los que se ha compartido el tablero de trabajo.

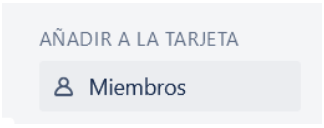


Fig. 6.51 Añadir miembros

En el apartado de etiquetas se despliega un menú que permite asignar un color a la tarjeta que se está creando, asignar colores permite catalogar el avance o urgencia de las tarjetas, por ejemplo, el rojo puede ser usado como indicador de que se debe solucionar de manera urgente el amarillo como prioridad intermedia y el verde como un cambio que no es vital para continuar con el desarrollo y puede esperar para dar solución.

Las checklist permiten añadir una cantidad de tareas pequeñas o pendientes dentro de la tarjeta, esto puede ayudarnos a escribir paso a paso la construcción de la solución al problema presentado en la tarjeta.

También permite añadir fechas tanto de inicio como de termino de cada tarjeta, esto añade información importante sobre el avance en la solución, así como tiempos limites que deben cumplirse.

La sección de adjunto sirve para añadir documentación útil que permita comprender la solución de manera más sencilla se pueden agregar enlaces, o conectarse con servicios de alojamiento como Dropbox.

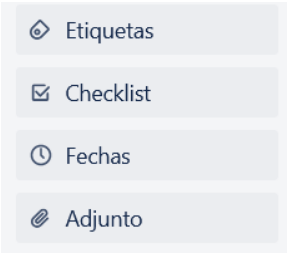


Fig. 6.52 Características de la nueva tarjeta

Una vez creadas y personalizadas las tarjetas es posible moverlas entre las diferentes columnas o listas donde se puede ver de manera general el actual estado de dicha tarjeta, si se encuentra en proceso, o si ha sido terminada y está lista para una nueva ejecución de pruebas.

6.4.2 Directrices para pruebas

Para la definición de pruebas es importante identificar el comportamiento que se espera ver en el software, así como los llamados flujos alternos, que ocurren cuando se ejecutan procesos extras para tratar la información de entrada y generar una salida adecuada, estos flujos alternos son causados por posibles errores de ejecución.

A este conjunto de condiciones y características que ayudan a determinar si el comportamiento del sistema cumple con lo esperado se le conoce formalmente como *caso de prueba,* incluyen una descripción de la funcionalidad que se probará, en estos se describen las entradas conocidas o datos que ingresan a la sección del software y la salida esperada que es el resultado de la ejecución del código.

La creación de estos casos de prueba debe ser redactados previos a cualquier ejecución de pruebas, ya que nos permitirán conocer lo que se espera ver cuando se ejecuta el sistema o parte de este.

Las directrices que se describen a continuación tienen como finalidad ayudar a los involucrados a redactar cada uno de los tipos de pruebas que deben ser ejecutadas sobre el software desarrollado, para esto pueden tomar como apoyo las plantillas diseñadas para la documentación de cada caso de prueba.

Definición de pruebas unitarias

Las pruebas unitarias se realizan en piezas muy pequeñas de software, procesos cortos o tareas simples, su finalidad es verificar el correcto funcionamiento del software en etapas muy tempranas de su desarrollo.

1.1 En la definición de pruebas unitarias se debe asignar un identificador del requerimiento que se va a probar de esta forma es sencillo relacionar el requerimiento descrito junto con la prueba que se va a ejecutar.

1.2 Los datos de entrada son los que se necesitan establecer previo a la ejecución del proceso es decir si el proceso es una suma de dos números enteros entonces los datos de entrada pueden ser cualquier dato que cumpla con las características propuestas.

1.3 Los datos de salida son lo que se espera ver, describir el formato de salida que esperamos ver y dependiendo de los datos de entrada se pueden determinar los datos de salida, siguiendo el ejemplo anterior se espera ver como salida un numero entero resultado de la suma de los dos datos de entrada.

1.4 El flujo de ejecución puede ser descrito como los pasos que se ejecutan en el requerimiento, si se debe realizar una consulta a base de datos, modificar un número, sumar, restar, llamar a alguna otra función del programa etc.

1.5 El flujo alternativo es aquel en que el sistema debe seguir en caso de alguna excepción, por ejemplo, si se esperaba un numero como dato de entrada y se ingresa una letra o un símbolo, si surge algún error en la base de datos etc.

1.6 Las funciones involucradas dentro del flujo de ejecución esperado, si es que este proceso llama a alguna funcionalidad o proceso puede ser una externa que no está construida específicamente para el requerimiento probado o una funcionalidad desarrollada para el requerimiento.

1.7 Observaciones en esa sección se pueden añadir los comportamientos inusuales encontrados o nuevas características, si es el caso.

1.8 Las sugerencias de correcciones van enfocadas a como se pueden solucionar los problemas o como se pueden construir las posibles mejoras.

1.9 Comisionado es el encargado de la ejecución de dicha prueba, únicamente quien ejecuta la prueba y redacta la prueba.

Se propone llenar con estos datos la plantilla sugerida para documentar los casos de pruebas unitarias:

Definición de pruebas unitarias		
Id prueba *[nombre asignado a la prueba, puede ser el mismo que el requerimiento que va a ser probado]*	Datos de entrada *[datos que deberán ser procesados por el software]*	Datos de salida *[datos que se esperan ver como resultado del proceso]*
Flujo de ejecución esperado *[secuencia de pasos que deben ejecutarse durante el proceso, pueden ser las funciones ejecutadas]*		
Flujo de operación alternativa *[procesos que se ejecutan cuando es necesario realizar algún paso extra al normal o cuando surgen excepciones en el flujo esperado]*		
Funciones o métodos llamados *[se describen los requerimientos de software que serán probados]*		
Observaciones *[notas sobre el comportamiento del software durante la prueba]*		
Sugerencia de correcciones *[cuando es necesario modificar o corregir el software se añade una descripción de la solución que deberá ser aplicada]*		
Comisionado *[Responsable a cargo de ejecutar la prueba]*		

Fig. 6.53 Plantilla Definición de pruebas unitarias

Definición de pruebas de integración

Estas pruebas se ejecutan después de las pruebas unitarias, una vez que se han construido módulos de software más complejos, con la finalidad de verificar que las piezas unitarias funcionan correctamente cuando interactúan con otras partes del software.

1.1 Generar un identificador de la prueba de integración, un nombre con el que referirse a la prueba

1.2 Los datos de entrada son los datos que se ingresan a la porción de sistema que se está probando, los datos de salida es el resultado que se espera ver después de pasar por una serie de funciones y procedimientos.

1.3 Flujo que se espera ejecutar de manera normal, una secuencia de procesos sobre los datos de entrada.

1.4 Operaciones alternativas o excepciones los diferentes caminos que pueden ejecutarse cuando ocurre alguna excepción o los datos deben recibir un tratamiento alterno.

1.5 Los módulos de software son el conjunto de funciones involucrados en una prueba de integración, es decir todas las partes del sistema que han sido probados individualmente están involucrados en esta prueba.

1.6 Observaciones en caso de que la prueba requiera volver a ser ejecutada, se encuentren puntos que deben revisarse o modificaciones a la porción del software, estas observaciones pueden ser la descripción de que ocurre con el sistema al ser probado como un conjunto.

1.7 Sugerencias de correcciones, en caso de que sea necesario modificar algo dentro del sistema, en esta sección es donde se debe detallar las posibles mejoras

1.8 Definir al responsable de ejecutar la prueba de regresión

Se propone llenar con estos datos la plantilla para generar los casos de pruebas de integración.

Definición de pruebas de integración

Id prueba de integración *[nombre asignado a la prueba, debe ser sencillo de recordar para los involucrados]*	
Datos de entrada *[datos que deberán ser procesados por el software]*	Datos de salida *[datos que se esperan ver como resultado del proceso]*
Flujo de ejecución esperado *[secuencia de pasos que deben ejecutarse durante el proceso, pueden ser las funciones ejecutadas]*	
Flujo de operación alternativa *[procesos que se ejecutan cuando es necesario realizar algún paso extra al normal o cuando surgen excepciones en el flujo esperado]*	
Módulos de software ejecutados *[se describen las piezas individuales de software que forman parte de la integración]*	
Observaciones *[notas sobre el comportamiento del software durante la prueba]*	
Sugerencia de correcciones *[cuando es necesario modificar o corregir el software se añade una descripción de la solución que deberá ser aplicada]*	
Comisionado *[Responsable a cargo de ejecutar la prueba]*	

Fig. 6.54 Plantilla Definición de pruebas de integración

Definición de pruebas de regresión

Las pruebas de regresión son la ejecución de pruebas anteriores con la finalidad de rectificar el correcto funcionamiento del software después de la corrección de algún error. Los desarrolladores pueden tomar la plantilla como punto de partida para documentar sus pruebas de regresión.

1.9 Identificador de la prueba que se probara nuevamente, es decir si es una prueba unitaria entonces se tomara el identificador de esa prueba, ocurre lo mismo en el caso de una prueba de integración se asigna el id en este campo de la plantilla definición de pruebas de regresión.

1.10 Que prueba se está ejecutando nuevamente, puede ser una prueba unitaria o de integración.

1.11 Las pruebas pueden ser ejecutadas más de una vez sobre una misma pieza de software, por lo que esta sección es un contador de cuantas veces se a ejecutado nuevamente una prueba.

1.12 Los campos llamados Entrada y Salida pueden ser tomados de la prueba que se está volviendo a ejecutar

1.13 Lo mismo ocurre con los campos de flujos de ejecución esperado y alternativo, contienen la misma información correspondiente a la prueba que se volverá a ejecutar como prueba de regresión.

1.14 Observaciones si en este nuevo ciclo de pruebas ocurre un error inesperado o surge nueva información que anteriormente no había aparecido, entonces se describirá en este campo que ocurrió cuando la prueba se volvió a ejecutar y como se comportó el sistema.

La siguiente plantilla permite a los involucrados dar seguimiento a las pruebas de regresión, en donde se añaden los datos de las pruebas que se ejecutaran nuevamente, para comprobar que el funcionamiento del software siga siendo el adecuado.

Definición de pruebas de regresión

Identificador *[identificador de la prueba que se repite]*					
Tipo de prueba *[unitaria o de integración]*					
No. De ejecución sobre esta prueba *[contador de ejecución]*					
Entradas *[datos que deberán ser procesados por el software]*					
Salidas *[datos que se esperan ver como resultado del proceso]*					
Flujo de ejecución esperado *[secuencia de pasos que deben ejecutarse durante el proceso, pueden ser las funciones ejecutadas]*					
Flujo de ejecución alternativo *[se ejecutan cuando es necesario realizar algún paso extra al normal o cuando surgen excepciones en el flujo esperado]*					
Observación *[notas sobre el comportamiento del software durante la prueba]*					
Comisionado *[Responsable a cargo de ejecutar la prueba]*					

Fig. 6.55 Plantilla Definición de pruebas de regresión

Gestión de pruebas

Posterior a la ejecución de pruebas cuando ya se tienen descritas las observaciones de cada una, así como las propuestas de solución, es recomendable utilizar la herramienta de gestión Trello que permite añadir tarjetas con una descripción del problema, así como checklist con los pasos a seguir para construir una solución y de ser necesario añadir documentación más específica de cómo funciona el proceso que dará solución a los istes encontrados.

Una buena práctica en Trello consiste en añadir un responsable de dar respuesta al problema encontrado, así como catalogar la tarjeta de acuerdo con su nivel de prioridad y asignar tiempos para no retrasar el resto del desarrollo.

6.5 Mantenimiento y evolución

El software no se degrada de forma física como lo hace el hardware, este se encuentra en constante evolución lo que deriva en un proceso de mantenimiento que puede resultar complejo, ya que involucra nuevos requerimientos o modificaciones al software original con la finalidad de adaptarlo a las nuevas necesidades de los usuarios.

Dichos cambios son parte del mantenimiento o evolución al software que es una actividad dentro del *ciclo de vida* del software y ocurre posterior a la entrega del sistema, consiste en fortalecer el sistema, comprobar el rendimiento y gestionar posibles mejoras, un buen mantenimiento de software no solo modifica y corrige, también permite que se conozca el estado y las condiciones actuales del funcionamiento.

El mantenimiento de software se considera como un nuevo ciclo en el desarrollo evolutivo además de los utilizados para la creación del software, dar mantenimiento a un software incrementa sus funcionalidades, las adapta o las mejora, este ciclo permite que el sistema siga evolucionando y no se vuelva obsoleto.

6.5.1 Herramientas para el mantenimiento o evolución

Adaptar un software a un entorno operacional resulta una tarea menos compleja cuando se tiene la documentación adecuada y una herramienta que permita gestionar dichos cambios, versiones etc. Las matrices de trazabilidad permiten seguir el proceso de evolución de un sistema de software en cada una de sus etapas, debe darse desde la etapa de descripción de requerimientos hasta la implementación del sistema y viceversa.

El uso de esta matriz permite a los investigadores “rastrear” un requerimiento desde su definición, codificación y pruebas, hasta su término. La matriz de trazabilidad permite relacionar cada requerimiento con su correspondiente entregable, ayuda a documentar diferentes aspectos del software como la funcionalidad deseada con los módulos del sistema, los componentes, las pruebas en las que está implicada.

En una matriz de trazabilidad las filas representan los requerimientos o funcionalidades del sistema y las columnas se representa algún aspecto de estos (componentes, pruebas o cualquier

sección de software a los que pertenece), cuando una funcionalidad esté relacionada con algún modulo del sistema se indica en la celda correspondiente de la matriz.

6.5.2 Directrices para manejar la evolución del software

La evolución en el software permite que su tiempo de vida continue sin dejar de cumplir con su propósito. Puede ser vista como los cambios en el código o la adición de nuevos procesos que deben ejecutarse. Para dar seguimiento a esta evolución se recomienda seguir las directrices que se encuentran en esta sección junto con el uso de la matriz de trazabilidad para su documentación.

1. Una vez que se han redactado los requerimientos de software es momento de comenzar a llenar la matriz de trazabilidad, que permite relacionar los módulos o piezas de software con cada requerimiento creado.
2. En la tabla permite relacionar los requerimientos con cada módulo o algún aspecto del sistema como componentes, pruebas, etc., estos módulos pueden ser nombrados de la misma forma que los paquetes en el diagrama.
3. Usar los identificadores asignados a cada requerimiento y añadirlos en la primera columna de la tabla.
4. Una vez que se han añadido los módulos y requerimientos es momento de relacionarlos, los requerimientos que son usados por algún módulo o se ejecutan dentro de un paquete deben ser marcados con una X.

Se propone llenar con estos datos la matriz de trazabilidad para documentar el seguimiento de la evolución:

	Modulo 1	Modulo 2	Modulo 3	Prueba 1	Prueba 2	Prueba 3	BaseDeDatos 1	BaseDeDatos 2
REQ1								
REQ2								
REQ3								
REQ4								
REQ5								

Fig. 6.56 Matriz de trazabilidad

Capítulo 7. Validación de propuesta por expertos y resultados obtenidos

Con la finalidad de conocer la opinión de expertos que han desarrollado Software Científico en sus investigaciones, se generó un cuestionario de evaluación aplicado a 2 estudiantes del posgrado de ciencias e ingeniería en computación y a 3 investigadores.

Para este fin se desarrolló un cuestionario en donde se describen las directrices su objetivo dentro del desarrollo de Software Científico y como deben ser utilizadas las plantillas de documentación y posteriormente se encuentra una serie de preguntas diseñadas para evaluar las directrices propuestas.

Las preguntas propuestas en el cuestionario de evaluación han sido diseñadas para validar la utilidad de las directrices para el desarrollo de Software Científico. El cuestionario como se entregó a los expertos se encuentra el Anexo 2.

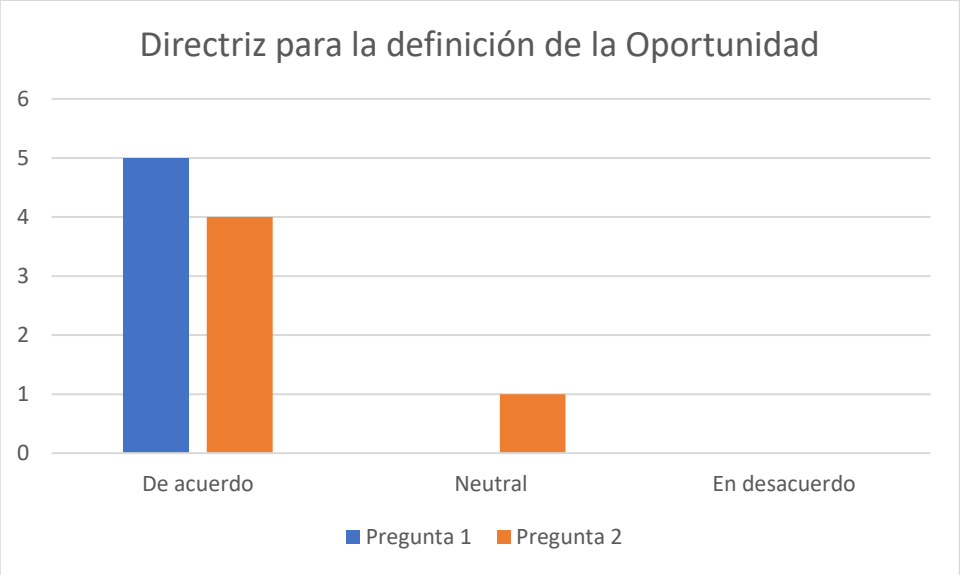
Respuestas al cuestionario para la validación

El cuestionario con las directrices propuestas ha sido respondido por diferentes expertos y alumnos del posgrado de Ciencias e Ingeniería en Computación. Las preguntas se encuentran numeradas en las siguientes secciones, cada sección corresponde a una directriz y cada tabla corresponde a las respuestas de validación por parte de los expertos.

Directriz para la definición de la Oportunidad

Las directrices sobre la Oportunidad buscar ayudar a los involucrados a definir la utilidad o ventaja que representa desarrollar el software, cuando la oportunidad es comprendida por cada uno de los involucrados se logra entender la razón detrás de cada decisión tomada.

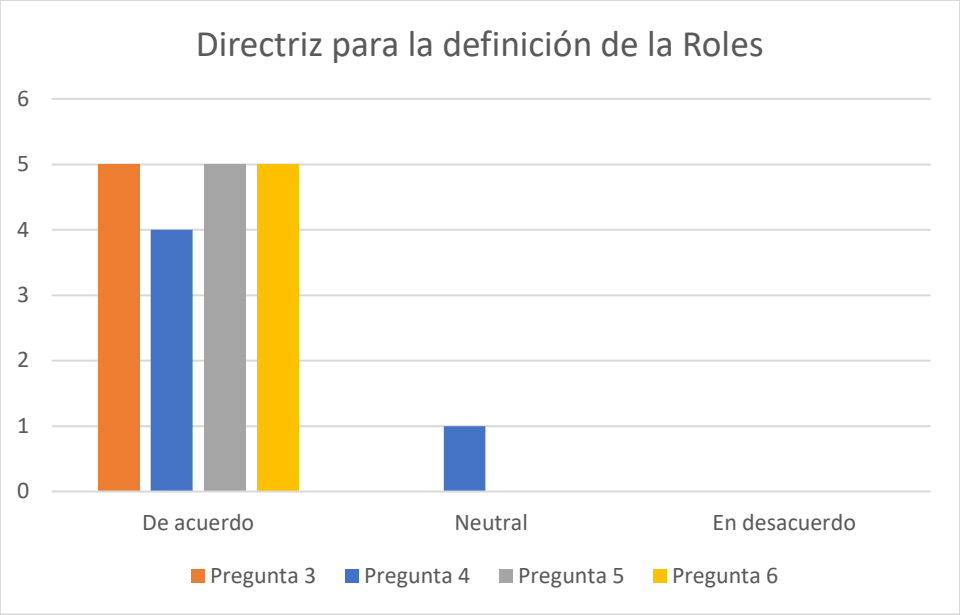
		De acuerdo	*Neutral*	*En desacuerdo*
1	*Definir la oportunidad ayuda a entender la idea tras el proyecto de desarrollo*			
2	*Comprender la oportunidad es posible identificar características importantes del desarrollo Determinar la viabilidad del proyecto es importante*			



Directriz para la definición de la Roles

En la definición de roles de la propuesta se espera que los involucrados se familiaricen con los roles y las responsabilidades que estos deben cumplir, de forma que sea sencillo comprender que actividades corresponden a cada rol, tomando en cuenta que un involucrado puede realizar más de un rol.

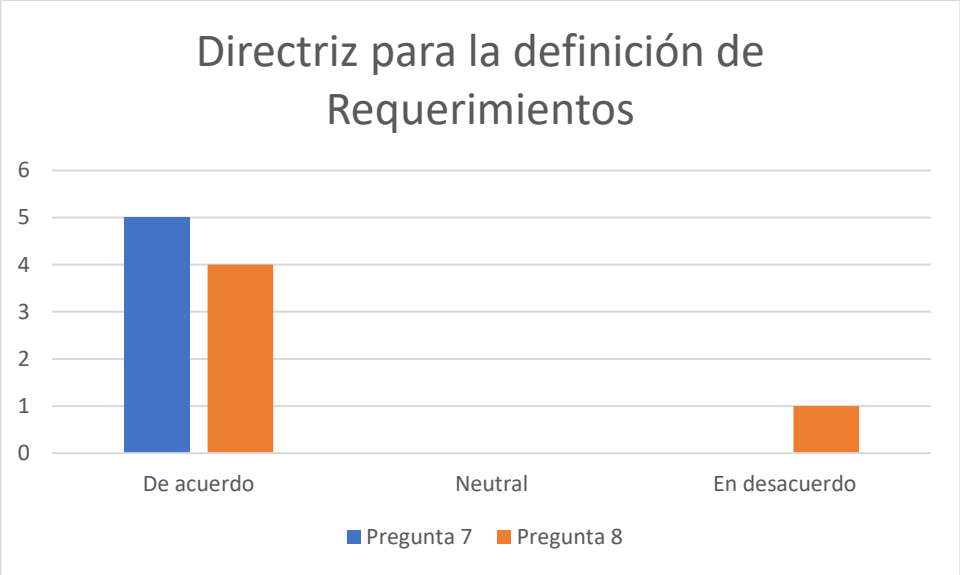
		De acuerdo	*Neutral*	*En desacuerdo*
3	*Definir roles para el desarrollo del software es importante*			
4	*Definir responsabilidades para cada rol de los involucrado antes de comenzar al desarrollo, es una práctica útil*			
5	*Las definiciones de los roles propuestos permitirán a los involucrados entender mejor como apoyar a que el desarrollo avance*			
6	*Es claro que en algunos casos un involucrado puede desempeñar más de un rol*			



Directriz para la definición de Requerimientos

El uso de un formato para cada requerimiento permitirá a los desarrolladores mantener la misma información por lo que la plantilla propuesta resulta una herramienta útil para evitar omitir información.

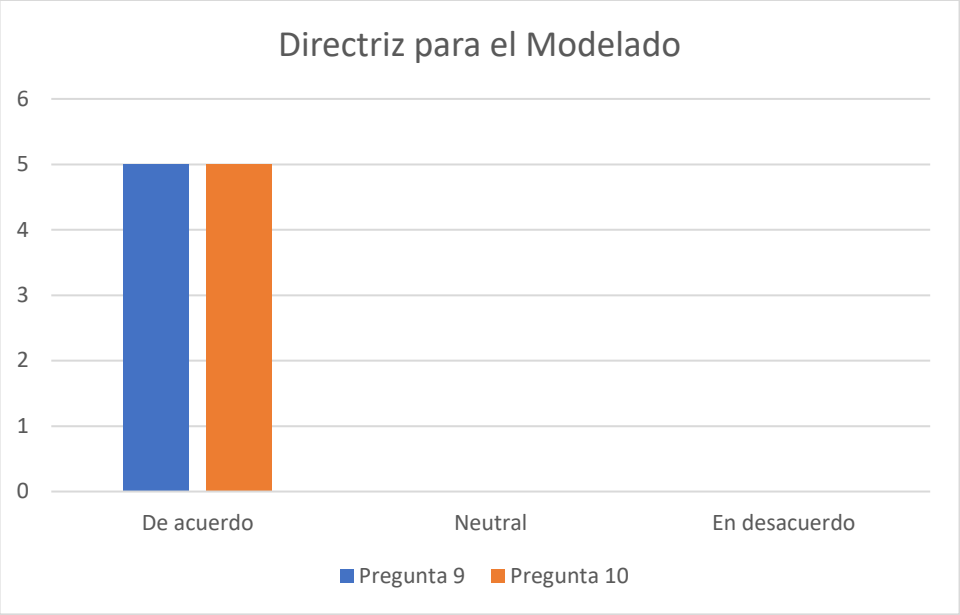
		De acuerdo	*Neutral*	*En desacuerdo*
7	*Contar con un formato para todos los requerimientos facilita la redacción e incluye datos importantes para todos los requerimientos*			
8	*La priorización de requerimientos es una práctica útil para determinar con que requerimientos se debe comenzar, los criterios de aceptación dan una idea del funcionamiento esperado por lo que es una sección fundamental en un requerimiento*			



Directriz para el Modelado

La directriz propuesta para el modelado del sistema a partir de un diagrama de paquetes tiene como objetivo tener una imagen de las interacciones que hay entre cada parte del sistema.

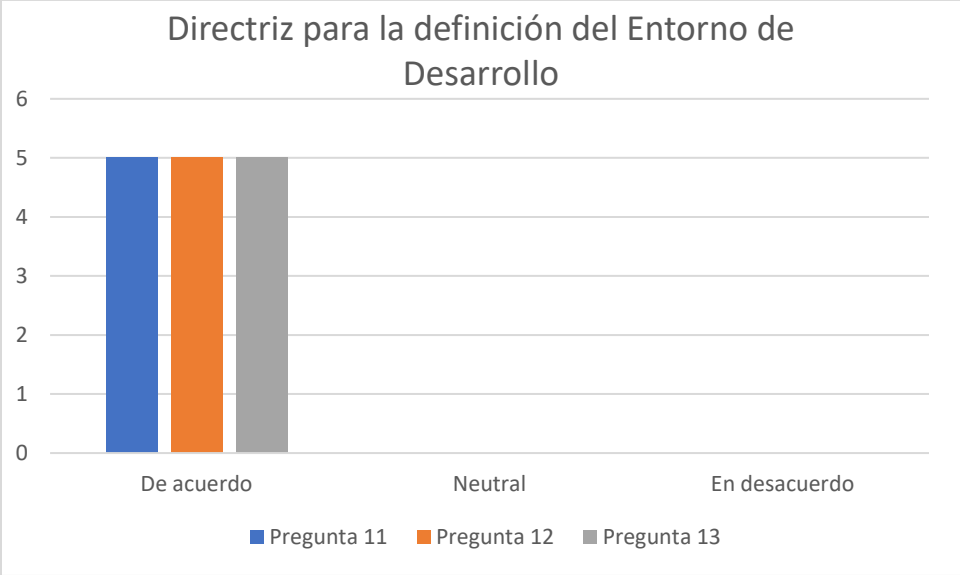
		De acuerdo	*Neutral*	*En desacuerdo*
9	*Modelar un diagrama de paquetes permite visualizar mejor la comunicación entre las partes del software*			
10	*Utilizar notas para establecer las entradas y salidas de algunos paquetes da información útil a los desarrolladores*			



Directriz para la definición del Entorno de Desarrollo

Definir todas las tecnologías a partir de la tabla propuesta que permite a los involucrados aprender a seleccionar la mejor opción de lenguaje de programación, gestor de bases de datos etc.

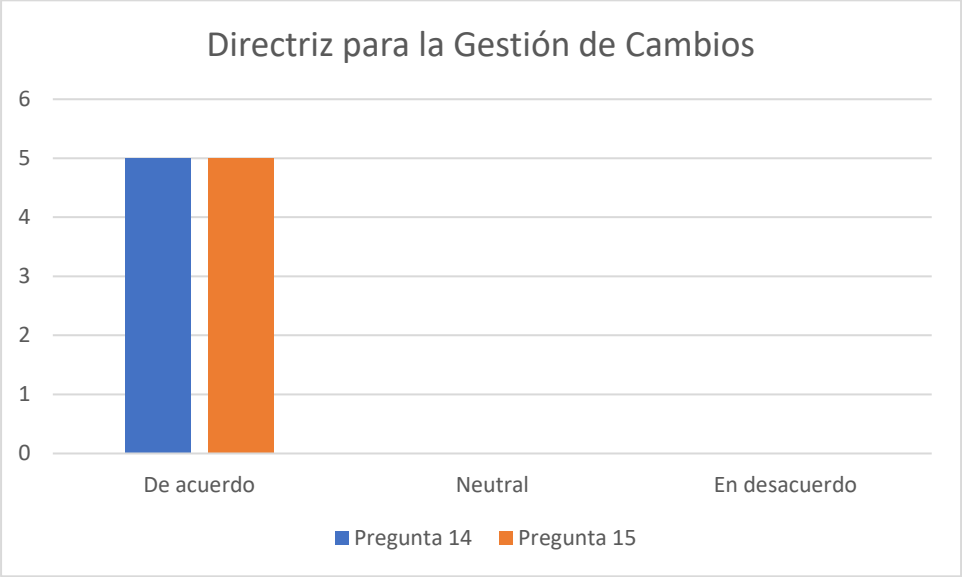
		De acuerdo	*Neutral*	*En desacuerdo*
11	*Definir las tecnologías y versiones que serán utilizadas evita confusiones cuando los desarrolladores preparen su entorno de trabajo*			
12	*Determinar las características técnicas del software ayuda a seleccionar las tecnologías para el desarrollo*			
13	*Las tecnologías dependen de factores como el conocimiento de los desarrolladores y las características técnicas del software*			



Directriz para la Gestión de Cambios

Mantener una gestión adecuada dentro del proyecto ayuda a los involucrados a conocer el avance y cambios en el código, así como llevar un registro de las diferentes versiones.

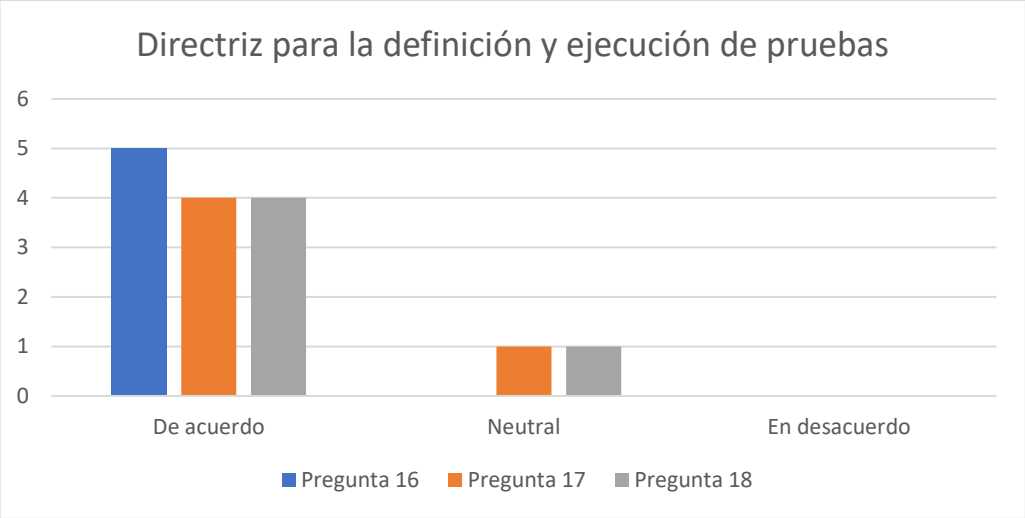
		De acuerdo	*Neutral*	*En desacuerdo*
14	*Gestionar de manera adecuada los cambios en el desarrollo es una práctica útil cuando se trata de un software que debe ser ajustado de manera constante, un repositorio de versionamiento permite tener control sobre los avances y cambios realizados durante el desarrollo*			
15	*Realizar una fusión del código de diferentes desarrolladores es un proceso importante que puede ser ejecutado con ayuda de un gestor de versionamiento.*			



Directriz para la definición y ejecución de pruebas

La directriz propuesta para la definición y ejecución de pruebas tiene como objetivo una constante revisión del funcionamiento del software, evitando detectar errores graves durante la etapa de producción.

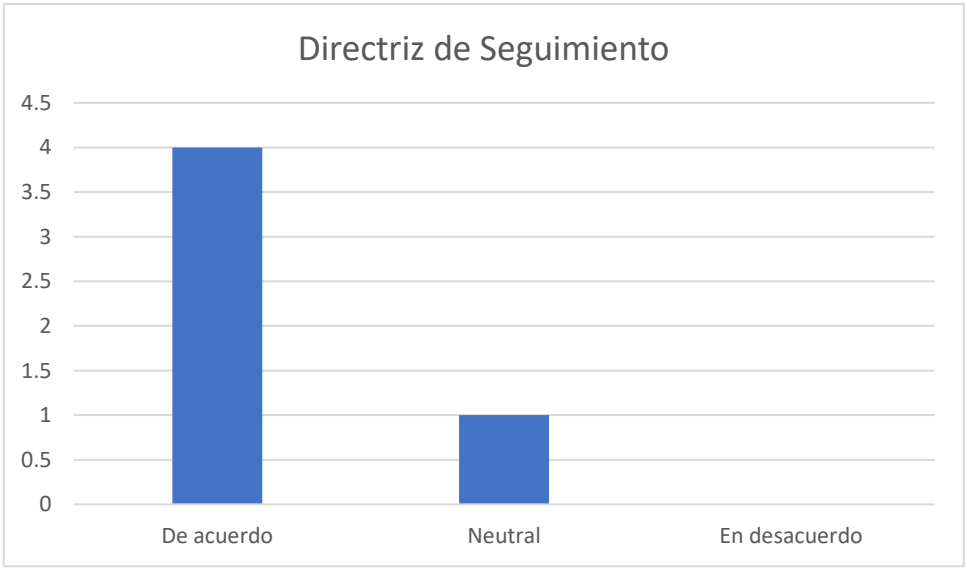
		De acuerdo	*Neutral*	*En desacuerdo*
16	*La ejecución de pruebas en etapas tempranas del desarrollo ayuda a encontrar errores en el código y resolverlos antes de que sea complicado identificar el origen*			
17	*Las pruebas son parte de la documentación que respalda la funcionalidad del software por lo que utilizar plantillas para la definición de pruebas unitarias ayuda a homogenizar la información en la redacción de cada prueba*			
18	*Las directrices anteriores son de ayuda para complementar la información de las pruebas*			



Directriz de Seguimiento

Finalmente se propone una directriz que permite a los investigadores dar seguimiento a cada requerimiento descrito en las plantillas anteriores, ver las modificaciones a través del ciclo del desarrollo.

		De acuerdo	*Neutral*	*En desacuerdo*
19	*Dar seguimiento a los requerimientos ayuda a identificar los cambios realizados dentro del software, conocer las interacciones entre los procesos del software permite conocer su flujo de ejecución esperado*			



Comentarios de los expertos

1. Actualmente existen cientos de herramientas y bibliotecas de software que aligeran el desarrollo de un nuevo software que será de utilidad en la obtención de los resultados de una investigación científica, es decir no es obligatorio desarrollar un software desde cero. Sin embargo, muchas de las directrices propuestas son herramientas útiles, aunque se utilice un software prediseñado por ejemplo determinar roles entre los involucrados.

2. Al igual que en todo software existen un sinfín de particularidades en el desarrollo, ya sea en los requerimientos de cada software, el número de involucrados, el tiempo, el presupuesto etc., por lo que puede darse el caso en el cual los cambios durante el desarrollo puedan cambiar la oportunidad que ha sido establecida al inicio del proyecto. Al tratarse de un desarrollo que avanza en ciclos es posible adaptar la construcción del software para alinearla con los cambios en la oportunidad

3. Las bibliotecas que automatizan las pruebas unitarias sin duda son una herramienta sumamente útil para agilizar procesos, reduciendo errores de llenado y evitan el llenado manual de documentos. Estas herramientas sin duda son un gran apoyo cuando se realiza un gran volumen de pruebas sobre el software desarrollado, pero pueden resultar en un contratiempo si uno o varios involucrados desconocen el funcionamiento de dichas herramientas, relacionar los requerimientos con los módulos es una propuesta sumamente útil para identificar que requerimientos están involucrados con una sección del software en específico.

4. En el caso de las pruebas pueden ser modificadas a este software ya diseñado, llamándolas pruebas de calibración, estas pruebas tendrán el objetivo de verificar la adecuada interacción entre los datos de investigación y el resultado obtenido por este software, pues generalmente en estos no se utilizan gestores de bases de datos.

En conclusión, de acuerdo con los comentarios hechos por parte de expertos la propuesta de directrices presentada es un modelo sencillo para ayudar a la creación de software de calidad en proyectos científicos, manejando conceptos que cualquier desarrollador está familiarizado y en algunos casos, en las investigaciones científicas no son aplicadas como deberían, sin embargo, los involucrados en este tipo de desarrollo podrían beneficiarse de las propuestas realizadas en este trabajo

Conclusiones.

La forma en cómo se desarrolla Software Científico se ha ido adaptando a través de los años tomando algunas prácticas de estándares y metodologías desarrolladas por la industria, sin embargo, hay características que deben llevarse con especial detalle cuando se construye software en el ámbito científico, ya que es diferente la forma en que se desarrolla un software comercial y un software que permita a los investigadores obtener resultados precisos dentro de sus proyectos.

El uso de prácticas de Ingeniería de Software en etapas tempranas del desarrollo de Software Científico puede permitir identificar posibles retos antes de comenzar a codificar, define el alcance del desarrollo y establece las funcionalidades más importantes que deben ser codificadas por los programadores.

Las directrices descritas dentro de este trabajo permiten a los involucrados incluir prácticas de ingeniería de Software en sus desarrollos y pruebas, generando documentación de calidad que permita respaldar los resultados obtenidos en la investigación y que puede ayudar a más investigadores a crear nuevos métodos que puedan dar solución o mejorar las ya existentes.

Las plantillas propuestas permitirán a los interesados disponer de estructuras y herramientas que pueden utilizar y adaptar para la definición de requerimientos de software, el modelado del sistema, determinar el ambiente de trabajo, llevar una adecuada gestión de versiones utilizando plataformas especializadas y dar seguimiento de la definición, ejecución de pruebas.

Trabajo futuro

Dado el tiempo limitado al desarrollar este trabajo y la situación que se ha llevado en el último año han dificultado la posibilidad de apoyar a investigadores en la adopción de estas directrices propuestas durante el desarrollo de su Software Científico, y se espera que estas directrices sean utilizadas pronto en algún proyecto desarrollo de Software Científico. Se espera que en los semestres posteriores dentro del posgrado estas directrices sean utilizadas por los investigadores, tutores y alumnos que se encuentren desarrollando software en algún proyecto de investigación.

Otro objetivo del trabajo a futuro es incluir nuevas directrices para complementar o mejorar las ya creadas en este trabajo, siempre y cuando estas también tengan bases de ingeniería de software, así como el respaldo de expertos en el área. Estas nuevas directrices podrían surgir después de que un grupo de investigadores utilizaran la propuesta de este trabajo y detectaran posibles mejoras o nuevas necesidades de directrices en el desarrollo de su proyecto.

Bibliografía

(s.f.).

Dai Clegg, R. B. (1994). *CASE Method Fast-track: A RAD Approach.* Addison-Wesley.

Github. (s.f.). Obtenido de https://conociendogithub.readthedocs.io/en/latest/data/dinamica-de-uso/

Hinsen, K. (2013). "*Software Development for Reproducible Research,Writing Programs for Scientists*" in Computing in Science & Engineering. 60-63.

Improvement, I. f. (2004). *Idea Generation Tools: Brainstorming, Affinity Grouping, and Multivoting.*

ISO/IEC. (2003). *Information Technology — Process Assessment.* Geneva, CH, Switzerland.

ISO/IEC. (2008). *Software Life Cycle Processes.*

ISO/IEC. (2011). *Lifecycle profiles for Very Small Entities.*

Kelly, G. (1995). *The Psychology of Personal Constructs.*

Lehman, M. (1980). "*Programs, life cycles, and laws of software evolution*".

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & W. (1997). "Metrics and laws of software evolution—the nineties view" . *International Software Metrics Symposium*, 20-32.

Llope M., M. C.-B. (2017). *Interaction between top-down and bottom-up control in marine food webs.*

M. del Carmen Gómez, J. C. (2019). *Fundamentos de Ingeniería de Software*. Mexico: UAM.

Maller, P. (2012). "*New Practices to Structure and Elicit Improvement Opportunities in Scientific Software Development Teams*". *International Conference on Computational Science and Its Applications*, 121-124.

Martinez, E. (2012). Propuesta de Procedimiento para realizar pruebas de Caja Blanca a las aplicaciones que se desarrollan en lenguaje Python. *Rev. Colomb. de Computación*.

McGeorge, G. R. (1997). The Sorting Techniques: A Tutorial Paper on Card Sorts, Picture Sorts, and Item Sorts.

Mellon, U. C. (2018). *Capability Maturity Model Integration CMMI.*

MIT. (2016). *News on campus and around the world*. Obtenido de **https://news.mit.edu./2016/scene-at-mit-margaret-hamilton-apollo-code-0817/**

Moprosoft., Facultad de Contaduría Y Administración de la Universidad Nacional Autónoma de México (UNAM)., 2002.

OMG, o. m. (October de 2018). *Essence.* Obtenido de https://www.omg.org: https://www.omg.org/spec/Essence/1.2/PDF|

P Maller, A. S. (2012). New Practices to Structure and Elicit Improvement Opportunities in Scientific Software Development Teams.

P. Valenzuela-Toledo, C. C. (2018). "*Scientific Software Development: Qualitative background for a model*". *SCCC*, 1-8.

Pressman, R. (2000). *Software Engineering: A Practitioner's Approach* . McGrawHill.

U. Kanewala, J. B. (2014). *Testing scientific software: A systematic literature review.*

unam. (10 de Septiembre de 2008). *revista unam*. Obtenido de http://www.revista.unam.mx/vol.9/num9/art63/int63-1.htm

Anexos

Anexo 1. Cuestionario de exploración

- ¿Define en tus palabras que es el Software Científico?
- ¿Cuáles crees que serían las diferencias entre el Software Científico y el software comercial?
- ¿Actualmente sigues alguna metodología de Ing. de software en el desarrollo de tu software?
- ¿Cómo obtienes los requerimientos de tu software?
- ¿Quién toma la decisión de que funcionalidades son importantes en tu software?
- ¿Evalúas la calidad de tu software?
- ¿Con que tipo de pruebas?
- ¿Como llevas la documentación de tu desarrollo?

Anexo 2. Cuestionario de evaluación

- Definir la oportunidad ayuda a entender la idea tras el proyecto de desarrollo
- Comprender la oportunidad es posible identificar características importantes del desarrollo determinar la viabilidad del proyecto es importante
- Definir roles para el desarrollo del software es importante
- Definir responsabilidades para cada rol de los involucrado antes de comenzar al desarrollo, es una práctica útil
- Las definiciones de los roles propuestos permitirán a los involucrados entender mejor como apoyar a que el desarrollo avance
- Es claro que en algunos casos un involucrado puede desempeñar más de un rol
- Contar con un formato para todos los requerimientos facilita la redacción e incluye datos importantes para todos los requerimientos
- La priorización de requerimientos es una práctica útil para determinar con que requerimientos se debe comenzar, los criterios de aceptación dan una idea del funcionamiento esperado por lo que es una sección fundamental en un requerimiento
- Modelar un diagrama de paquetes permite visualizar mejor la comunicación entre las partes del software
- Utilizar notas para establecer las entradas y salidas de algunos paquetes da información útil a los desarrolladores
- Definir las tecnologías y versiones que serán utilizadas evita confusiones cuando los desarrolladores preparen su entorno de trabajo

- Determinar las características técnicas del software ayuda a seleccionar las tecnologías para el desarrollo
- Las tecnologías dependen de factores como el conocimiento de los desarrolladores y las características técnicas del software
- Gestionar de manera adecuada los cambios en el desarrollo es una práctica útil cuando se trata de un software que debe ser ajustado de manera constante, un repositorio de versionamiento permite tener control sobre los avances y cambios realizados durante el desarrollo
- Realizar una fusión del código de diferentes desarrolladores es un proceso importante que puede ser ejecutado con ayuda de un gestor de versionamiento.
- La ejecución de pruebas en etapas tempranas del desarrollo ayuda a encontrar errores en el código y resolverlos antes de que sea complicado identificar el origen
- Las pruebas son parte de la documentación que respalda la funcionalidad del software por lo que utilizar plantillas para la definición de pruebas unitarias ayuda a homogenizar la información en la redacción de cada prueba
- Las directrices anteriores son de ayuda para complementar la información de las pruebas
- Dar seguimiento a los requerimientos ayuda a identificar los cambios realizados dentro del software, conocer las interacciones entre los procesos del software permite conocer su flujo de ejecución esperado