PLAN DE ASEGURAMIENTO DE CALIDAD
28 DE ABRIL DE 2016

Kwan Tecnología
Calle 45 N. 351 Emiliano Zapata Oriente
C.P. 97144
01 999 988 9298

PREFACIO

Este documento contiene el Plan de Aseguramiento de la Calidad (SQA, por sus siglas en inglés) para la empresa Kwan. Este documento ha sido adaptado del documento SQA Plan Template, TM-SQA-01, v2.0.

REGISTRO DE CAMBIOS

Comentado [GPZ1]: Falta llenarlo

***A** – AÑADIDO **M** – MODIFICADO **D** – ELIMINADO

N. DE VERSIÓN	FECHA	N. DE FIGURA, TABLA O PÁRRAFO	**A* M D**	TÍTULO O DESCRIPCIÓN BREVE	N. DE PETICIÓN DE CAMBIO
1.0	28/02/16			Plantilla del documento	
1.1	12/03/16		A	Organigrama	
1.2	26/03/16		A	Procesos de la empresa	
2.0	16/04/16		M	Cambio de Plantilla	
2.1	19/04/16		A	Anexo de Propuestas	

TABLA DE CONTENIDOS

Sección

SECCIÓN 1. PROPÓSITO

El propósito de este plan es definir la organización de Aseguramiento de la Calidad de Software (SQA, por sus siglas en inglés) para la empresa Kwan, las tareas y responsabilidades de SQA; proveer documentos de referencia y guías para realizar las actividades de SQA; proveer estándares, prácticas y convenciones utilizadas al realizar las actividades de SQA; y proveer las herramientas, técnicas y metodologías para necesarias para apoyar a las actividades de SQA, con base en el IEEE Std. 730-1998, Standard for Software Quality Assurance Plans [1].

1.1 ALCANCE

Este plan establece las actividades de SQA realizadas durante el ciclo de vida de los proyectos de Kwan, enfocándose en las etapas de codificación y pruebas, no estando contempladas, por salirse del alcance del proyecto, las etapas relacionadas con el desarrollo de requisitos, diseño y el mantenimiento del producto de software.

El objetivo del plan SQA es verificar que todo el software y documentación entregable cumpla con los requisitos técnicos establecidos.

1.2 LISTA DE ACRÓNIMOS Y DEFINICIONES

A continuación, se muestra una lista de los acrónimos y definiciones usadas en este documento:

- **Back-end:** Parte del software que procesa la entrada desde el front-end.
- **Bug:** Error o fallo en un programa de computador o sistema de software que desencadena un resultado indeseado.
- **Framework:** Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software.
- **Front-end:** Parte del software que interactúa con el o los usuarios.
- **GP:** Gestión del Proyecto.
- **SQA:** Aseguramiento de la Calidad del Software.

1.3 VISTA GENERAL DEL DOCUMENTO

Este documento identifica la organización y procedimientos a ser usados para ejecutar las actividades relacionadas al plan de SQA como se especifica en IEEE Std. 730-1998, IEEE Standard for Software Quality Assurance Plans [1] y el IEEE Std 730.1-1995, IEEE Guide for SQA Planning [2]. A continuación, se muestra una vista general de las secciones del documento.

La sección 1 describe el propósito y contenido del plan SQA, las relaciones del plan con otros planes y lista todos los acrónimos y documentos referenciados en este plan.

La sección 2 describe cada elemento de la organización que influye en la calidad del software.

La sección 3 describe las tareas de SQA.

La sección 4 identifica los estándares, prácticas y convenciones.

La sección 5 describe el involucramiento del SQA en las pruebas.

La sección 6 describe el reporte de problemas y las acciones correctivas.

La sección 7 describe herramientas de SQA, técnicas y metodologías.

La sección 8 describe la herramienta de manejo de configuración usada para el control de código.

1.4 DOCUMENTOS DE REFERENCIA

[1] IEEE Std 730-1998, IEEE Standard for Software Quality Assurance Planning.

[2] IEEE Std 730.1-1995, IEEE Guide for Software Quality Assurance Planning.

[3] Modelo McCall – Criterios de calidad

[4] Modelo ISO-9126 – Criterios de calidad

[5] Modelo ISO-25000 – Criterios de calidad

[6] IEEE Std 1016 -1998 – Prácticas recomendadas para el diseño de software.

[7] IEEE Std 1008 - 1987 – Estándar para pruebas unitarias de software.

[8] IEEE Std 1061 - 1998 – Estándar para una metodología para la calidad de software.

[9] Kwan Tec – Plan de desarrollo de software.

[10] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications

[11] Jira – https://es.atlassian.com/software/jira – Herramienta utilizada para el seguimiento de proyectos de equipo de software.

[12] Mocha – https://mochajs.org/ – Framework de pruebas para JavaScript.

[13] Cordova – https://cordova.apache.org/ – Framework para el desarrollo de aplicaciones móviles.

[14] Protractor – http://angular.github.io/protractor/#/ – Framework de pruebas end-to-end para aplicaciones en AngularJS.

[15] Karma – https://karma-runner.github.io/0.13/index.html – Corredor de pruebas para JavaScript.

[16] W. Humphrey, "The Personal Software Process: Status and Trends," IEEE Software, Nov./Dec. 2000, pp. 71-75.

[17] J.M. Godoy, F. Gómez y E. Rubio," Análisis y Gestión del Desarrollo del Software", 2004.

[18] Lewis, W. and Veerapillai, G. (2005). "Software testing and continuous quality improvement.", Boca Raton: Auerbach Publications.

[19] "Estándares de Calidad de Sistemas Software: Modelo de Aseguramiento de la Calidad", Sociedad Informática del Gobierno Vasco.

[20] "Auditorias de Calidad para la mejorar la productividad", Dennis R. Arter, tercera edición, ASQ Quality Press.

[21] "Manual De Procesos Y Procedimientos De Auditoría Interna", Licda Nory Álvarez Betancourth, GUN.

[22] Radice, R.A.; Roth, N.K.; O'Hara, A.C., Jr; Ciarfella, W.A. " A Programming Process Architecture" IBM Systems Journal Vol.24, No 2 (1985), pp.79-90.

[23] Estándar de codificación basado en GNU.

SECCIÓN 2. GESTIÓN

Esta sección describe los elementos de la organización que tienen influencia en la calidad de software.

2.1 ORGANIZACIÓN

El encargado del área de gestión de calidad en el proyecto es el responsable de realizar la gestión que asegura que el proceso establecido sea realmente implementado y que los productos de ese proceso cumplan con los criterios de calidad establecidos en este plan. Las disciplinas de gestión brindan soporte a las disciplinas básicas (Requerimientos, Análisis, Diseño, Implementación, Implantación y Verificación) y se realizan en forma paralela a ellas. La Figura 2-1 muestra la organización del equipo de desarrollo.

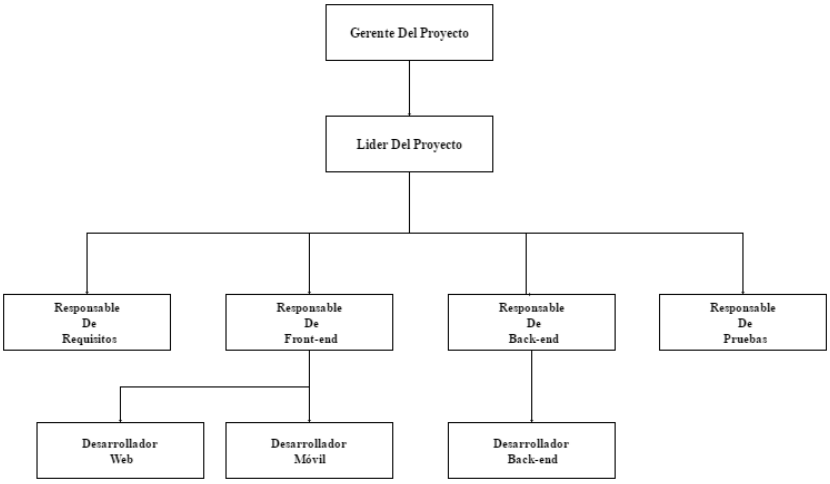

Figura 2-1. Organización del equipo de desarrollo

A continuación, se describen los grupos que influyen y controlan la calidad de software con sus respectivas responsabilidades.

a. Líder del proyecto es responsable de:
   - Informar al gerente del proyecto.
   - Administrar al equipo de trabajo.
   - Asignar las tareas.
   - Revisar productos.
   - Revisar procesos que se consideren relevantes para la empresa.
   - Manipulación de las bases de datos.
   - Configuración de los servidores locales.

b. Responsable de requisitos es responsable de:
   - Analizar las necesidades del cliente.
   - Identificar las posibles mejoras y cambios al producto, así como la agregación de nuevas características.
   - Dividir los requisitos en iteraciones.
   - Apoyar al líder del proyecto en la asignación de tareas en la herramienta Jira.

c. Líder de front-end es responsable de:

- Diseñar los aspectos gráficos de las páginas o aplicaciones tales como bosquejos y prototipos.
- Aceptar o rechazar los módulos o avances que entreguen los desarrolladores de front-end.
- Verificar aspectos básicos de un código mantenible tales como nombres correctos de variables, indentación y procedimientos en los métodos que soportan el nombre de estos para front-end.

d. Líder de back-end es responsable de:
   - Mantener las bases de datos funcionales.
   - Aceptar o rechazar los módulos o avances que entreguen los desarrolladores de back-end.
   - Verificar aspectos básicos de un código mantenible tales como nombres correctos de variables, indentación y procedimientos en los métodos que soportan el nombre de estos para back-end.

e. Desarrollador es responsable de:
   - Codificar las características específicas del área designada por el líder del proyecto.
   - Probar su propio código según sus propios criterios.

f. Responsable de pruebas es responsable de:
   - Diseñar el plan de pruebas.
   - Define patrones de aceptación.
   - Define casos de prueba.

2.2 RECURSOS

2.2 Instalaciones y equipamiento

El equipo de SQA tendrá acceso a las instalaciones y equipos definidos continuación para realizar funciones tales como evaluar los productos software, documentación y realizar las revisiones:

- 7 Monitores
- 3 Servidores
- Conexión a internet (inalámbrico o cableado)
- Fuente de corriente eléctrica
- Lápices
- Hojas de papel

2.2.2 Personal

El personal está familiarizado y es capaz de aplicar los estándares y pautas definidas en este documento, lo que implica el conocimiento previo de este plan. El personal también está familiarizado con los planes de desarrollo de software, plan de administración de la configuración, así como con las actividades relacionadas con el desarrollo de software, de diseño, codificación, pruebas y requerimientos. Es imprescindible que el personal tenga una cultura de trabajo en equipo. El personal requerido para las actividades de SQA son los siguientes:

- 1 Líder de proyecto
- 1 Responsable de requisitos
- 1 Líder de front-end
- 1 Líder de back-end
- 3 Desarrolladores
- 1 Responsable de pruebas
- 1 Gerente del proyecto

SECCIÓN 3. PROCESOS

En esta subsección se detallan los procesos que realiza Kwan actualmente, junto con las propuestas para introducir elementos de calidad a ellos.

3.1 OBTENCIÓN DE REQUISITOS

Este plan establece las actividades de SQA realizadas durante el ciclo de vida de los proyectos de Kwan, enfocándose en las

3.1.1 Propósito

Obtener las funcionalidades con las que el sistema de software contará.

3.1.2 Roles involucrados

- Gerente del proyecto.

3.1.3 Actividades

1. El gerente del proyecto contacta con el cliente vía correo electrónico para determinar la fecha y el lugar en el cual se reunirán.
2. El gerente del proyecto y el cliente se reúnen en la fecha y lugar acordados para dialogar las características del software.
3. El gerente del proyecto escucha las peticiones del cliente y toma apuntes informales durante la plática.

3.1.4 Criterios de salida

- Completar Lista de posibles requisitos.
- Completar Tickets en Jira.

3.2 ASIGNACIÓN DE TAREAS

3.2.1 Propósito

Asignar tickets con las tareas a cada desarrollador.

3.2.2 Roles involucrados

- Gerente del proyecto.
- Desarrollador (back-end o front-end).

3.2.3 Herramientas

- Jira.

3.2.4 Criterios de Entrada

- Finalizar Tickets con las tareas en Jira.

3.2.5 Actividades

1. El gerente del proyecto revisa los tickets en Jira para determinar a qué área pertenece el requisito (back-end o front-end).
2. El gerente del proyecto asigna las tareas en base a su experiencia y a las capacidades de cada desarrollador. Esta actividad es realizada en una junta semanal.

3. El desarrollador revisa las tareas de los tickets que se le hayan asignado. De tener duda, se consultará con el gerente del proyecto.
4. El gerente del proyecto junto con los desarrolladores involucrados acuerda los tiempos para la finalización y el alcance del requisito.

3.2. 6 Criterios de salida

- Finalizar la actualización de lista de requisitos.

3.3 PLANIFICACIÓN DE CODIFICACIÓN.

3.3.1 Propósito

Identificar características a codificar para definir un orden en el cual estas serán desarrolladas.

3.3.2 Roles involucrados

- Desarrollador (back-end o front-end).
- Líder de proyecto.

3.3.3 Herramientas

- Jira.
- Slack
- Plantilla de Registro de tiempos (véase Apéndice B, Tabla de Registro de Tiempos)..

3.3.4 Criterios de Entrada

- Tickets con las tareas a realizar hechos en Jira.
- Lista de asignación*.
  .

3.3.5 Actividades

1. El desarrollador crea una nueva entrada en el registro del Cuaderno de Registro de tiempos.
2. El desarrollador identifica cuáles son las tareas que se le han asignado, obteniendo una descripción de las funciones a programar.
3. El desarrollador ordena las tareas que se le han asignado de acuerdo a la prioridad que se le ha asignado en el ticket.

3.1. El desarrollador si llegara a encontrar dos o más ticket con la misma prioridad procede a comunicarse con el Líder de proyecto para definir un orden.

4. El desarrollador crea una lista con el orden en el que se van a desarrollar las características.
5. El desarrollador actualiza el registro correspondiente en su Cuaderno de Registro de Tiempos.

3.3. 6 Criterios de salida

- Lista de características a desarrollar con prioridad asignada de acuerdo a cada desarrollador.

3.4 CODIFICACION.

3.4.1 Propósito

Desarrollar las características identificadas usando las herramientas a elección de cada desarrollador.

3.4.2 Roles involucrados

- Líder del proyecto.
- Desarrollador (back-end o front-end).

3.4.3 Herramientas utilizadas

- IDE's.: NetBeans, Intellij IDEA, Eclipse
- Framework's. : Cordova
- GitHub.
- Slack.
- E-mail.
- Cuaderno de Registro de tiempos (véase Apéndice B, Tabla de Registro de Tiempos).

Comentado [AA2]: Especificar qué frameworks y qué IDEs o hacer referencia a la lista de los mismos

Comentado [GPZ3R2]: Ya esta

3.4.4 Criterios de entrada

- Tickets con las tareas a realizar en Jira.
- Lista ordenada de características a desarrollar con prioridad asignada de acuerdo a cada desarrollador.

3.4.5 Actividades

1. El desarrollador crea una nueva entrada en el registro del Cuaderno de Registro de tiempos.
2. El desarrollador desarrolla módulos atómicos, de forma que un módulo sea una funcionalidad.
    a. El líder del proyecto resuelve problemas generados durante la codificación, se contacta con este según la complejidad del problema (ver la Tabla 2-2).
3. El desarrollador al terminar un módulo procede a realizar las pruebas correspondientes (ver 3.5).
4. El desarrollador marca el ticket como completado
5. El desarrollador actualiza su rama de codificación (ver Sección 9).
6. El desarrollador comunica al líder de su área el avance del proyecto.
7. El líder del proyecto verifica los avances y actualiza la rama principal.
    a. El líder del proyecto valida la entrada si no existe ningún error o defecto
8. El desarrollador anota el tiempo de codificación (véase Apéndice B, Tabla de Registro de Tiempos).

Prioridad	Descripción
Alta	El problema no permite que se siga codificando, se contacta al Líder del proyecto en persona de estar presente o vía telefónica de no estarlo
Media	El problema aumenta el tiempo de la codificación. Se contacta al líder del proyecto vía chat de Slack.
Baja	Se puede seguir codificando aun con el problema existente, se contacta al líder del proyecto de ser necesario vía e-mail.

Tabla 2-2.Tabla de prioridades [18]

3.4.6 Criterios de salida

- Tickets marcados como completados.
- Tickets marcados como incompletos.
- Módulos codificados y verificados.

3.5 Pruebas Unitarias

3.5.1 Propósito

Verificación del código desarrollado.

3.5.2 Roles involucrados

- Tester.

3.5.3 Herramientas utilizadas

- Mocha.

3.5.4 Criterios de entrada

- Módulos o funcionalidades ya codificadas.
- Los tickets con las actividades a realizar que son definidas en la herramienta Jira.

3.5.5 Actividades

1. El tester analiza los requisitos con base en la complejidad de estos para determinar qué aspectos se van a probar.
    a. El tester debe tener en cuenta: elementos de todo el proyecto involucrados, prioridad del requisito, numero de características por requisito.
2. El tester utilizando el framework para pruebas Mocha define un conjunto de pruebas a las que se someterá el código. Dichas pruebas dependen de la característica y del programador.
3. El tester comienza a realizar las pruebas con ayuda del framework.
4. El tester analiza el resultado de la prueba.
    a. El tester al confirmar el fracaso de la prueba, debe analizar la lógica de la prueba. En caso de que la lógica de la prueba sea la correcta se procede a analizar el código para confirmar un defecto.
5. El tester registra el resultado de la prueba en el formato establecido(Apendice A, registro de Resultados de Prueba)

6. El tester actualiza los repositorios con los nuevos módulos o funcionalidades.

3.5.6 Criterios de salida

- Prototipo funcional.

3.6 PRUEBAS DE PORTABILIDAD

3.6.1 Propósito

Verificación del código desarrollado.

3.6.2 Roles involucrados

- Tester

3.6.3 Herramientas utilizadas

- Karma.

3.6.4 Criterios de entrada

- Módulos o funcionalidades ya codificadas.
- Los tickets con las actividades a realizar que son definidas en la herramienta Jira.

3.6.5 Actividades

1. El tester analiza los selecciona el modulo a ser probado, en base a la experiencia en trabajos previos.
2. El tester comienza a realizar las pruebas con ayuda del framework.
3. El tester analiza el resultado de la prueba.
    a. El tester al confirmar el resultado como un fracaso, debe registra el defecto.
4. El tester registra el resultado de la prueba en el formato establecido(Apendice A, registro de Resultados de Prueba)
5. El tester actualiza los repositorios con los nuevos módulos o funcionalidades.

Comentado [AA4]: O tester

Comentado [GPZ5R4]: corregido

3.6.6 Criterios de salida

- Prototipo funcional.

3.7 PRUEBAS END TO END

3.7.1 Propósito

Verificación del código desarrollado.

3.7.2 Roles involucrados

- Tester

3.7.3 Herramientas utilizadas

- Protractor.

3.7.4 Criterios de entrada

- Módulos o funcionalidades ya codificadas.
- Los tickets con las actividades a realizar que son definidas en la herramienta Jira.

3.7.5 Actividades

1. El tester analiza los requisitos con base en la complejidad de estos para determinar qué aspectos se van a probar.
    a. El tester debe tener en cuenta: elementos de todo el proyecto involucrados, prioridad del requisito, numero de características por requisito.
2. El tester comienza a realizar las pruebas con ayuda del framework.
3. El tester analiza el resultado de la prueba.
    a. El tester al confirmar el resultado de la prueba como un fracaso, debe registra el defecto.
4. El tester registra el resultado de la prueba en el formato establecido (Apendice A, registro de Resultados de Prueba)
5. El tester actualiza los repositorios con los nuevos módulos o funcionalidades.

Comentado [AA6]: O tester

Comentado [GPZ7R6]: corregido

3.7.6 Criterios de salida

- Prototipo funcional.

3.8 VERIFICACIÓN DEL ESTADO DEL PROYECTO

3.8.1Propósito

Verificación de los avances realizados.

3.8.2 Roles involucrados

- Gestor del Proyecto.

3.8.3 Criterios de entrada

- Tickets con las actividades a realizar hechos en Jira.
- Lista de asignación.
- Código desarrollado.
- Prototipo funcional.

3.8.4 Actividades

1. El GP analiza a que desarrollador se le asignó cada funcionalidad.
2. El GP determina con base en los tickets y en el código entregado si se cumplieron las metas de la iteración.
3. El GP determina:
4. El GP identifica anomalías y decide su nivel de estado crítico.
5. El GP asigna las nuevas tareas con base en los resultados mostrados.

3.8.5 Criterios de salida

- Lista de características a desarrollar.
- El producto de software validado.
- Lista de errores o situaciones sin resolver.

SECCIÓN 4. TAREAS DE SQA

Las tareas de SQA se realizan con relación a qué actividades de desarrollo de software se estén llevando a cabo. Una o más tareas de SQA pueden realizarse concurrentemente hasta que una tarea haya sido completada. Se le considera completa a una tarea cuando los reportes requeridos hayan sido satisfactoriamente completados o los ítems de acción hayan sido cerrados. Las siguientes tareas se realizarán como parte del rol de SQA y requerirán de la coordinación y cooperación de la mayoría del equipo.

4.1. TAREA: VERIFICACIÓN DEL PROYECTO, REVISIONES Y AUDITORIAS.

Definición de las revisiones y auditorías técnicas y de gestión que se realizarán. Especificación de cómo **SERÁN LLEVADAS A CABO DICHAS REVISIONES Y AUDITORÍAS.**

4.1.1 AUDITORIA.

Además de lo especificado en el Plan SQA, también contempla en general la realización de las siguientes auditoria:

• Auditoría de Código y Ejecución de Pruebas: El chequeo estático de código y las actividades de

Pruebas Unitarias, de Integración y de Sistema, son tareas de aseguramiento de calidad del producto software definidas y pautadas en la metodología [19].

• Auditorías de Fin de Fase: Al finalizar cada una de las fases que marca ARINBIDE se asegura que se hayan generado todos los productos obligatorios, y que se hayan seguido apropiadamente las actividades establecidas [19][20].

• Auditorías muéstrales: Si se considera necesario, teniendo en cuenta la complejidad del proyecto y los niveles de calidad que se vayan obteniendo en las actividades SQA realizadas, se contempla la realización durante el proyecto de verificaciones de documentación y/o ejecución de pruebas adicionales [19][20].

• Auditoría Final: coincide con el paso previo a producción y consolida los resultados de las actividades de calidad realizadas en el proyecto y el grado de calidad obtenido en los productos generados. Establece conclusiones y recomendaciones para la toma de decisiones en el paso a producción del producto [20] [19]. .

Una Auditoria estudia una actividad para verificar si fue realizado de acuerdo a las reglas, el análisis resultante dirá a las parte interesadas si la actividad fue realizada de acuerdo con los arreglos preestablecidos y si esos fueron los adecuados para lograr el resultado deseado.

La auditoría es un compromiso realizado por un auditor interno o externo, sin embargo, el planteamiento general de cada auditoría es el mismo.

La práctica de la Auditoría se divide en tres fases [20] [21]: Planeación, Ejecución, Informe.

SECCIÓN 5. ESTÁNDARES, PRÁCTICAS, CONVENCIONES Y MÉTRICAS

Para verificar la entrega de un producto de alta calidad, todo individuo asignado al proyecto participará en el aseguramiento de la calidad. La referencia [ ] define los procedimientos mediante los cuales el equipo de desarrollo verificará la calidad del producto durante el proceso de desarrollo. El resto de esta sección describe los procedimientos usados por SQA para verificar que las actividades de aseguramiento de la calidad de este plan y los estándares, prácticas, convenciones y métricas sean cumplidas.

SECCIÓN 6. PRUEBAS

Las actividades de pruebas realizadas en los proyectos de Kwan incluyen:

a. Pruebas unitarias
b. Pruebas de caja negra
c. Pruebas end-to-end
d. Pruebas de integración
e. Pruebas de portabilidad

El líder del proyecto designará a una persona como el responsable de pruebas, quien se encargará de elaborar un plan de pruebas para el proyecto que se desarrollará.

Todo personal que desarrolle algún ticket del proyecto realizará las siguientes actividades de pruebas necesarias del software:

- Realizar las pruebas unitarias.
- Realizar las pruebas de acuerdo al plan de pruebas.
- Documentar los resultados de las pruebas.
- Recomendar acciones correctivas si se encontraron defectos en los módulos probados.
- Enviar los reportes de las pruebas al responsable de pruebas.

SECCIÓN 7. REPORTE DE PROBLEMAS Y RESOLUCIÓN

En esta sección se describe el reporte y control del sistema utilizado por el personal de calidad para registrar y analizar las discrepancias encontradas, así como para monitorear la implementación de las acciones correctivas. Los formatos (ISO 19011, auditoria interna) utilizados para realizar los reportes se describen más adelante en esta sección.

7.1 REPORTE DEL PROCESO DE AUDITORÍA

El personal de calidad reportará el resultado de las auditorias y las recomendaciones proporcionadas. Este reporte se usa para asegurarse que el proceso:

1. Se está siguiendo de manera correcta y se está trabajando de forma efectiva
2. Se está siguiendo, pero no se está trabajando de manera efectiva
3. No se está siguiendo

7.1.1 Presentación del reporte del proceso de auditoria

El proceso de Reporte de Auditorias está dirigido hacia el Líder del Proyecto el cual utilizara los reportes de las siguientes maneras:

1. Para saber si los procesos de desarrollo son acatados y si son efectivos para el cumplimiento de las metas del proyecto. Cuando sea necesario el Líder del proyecto puede iniciar cambios a los procesos, mediante los procedimientos establecidos, para que los procesos queden estables.

2. Para indicar el acuerdo, desacuerdo, o el aplazamiento de las recomendaciones hechas en el Proceso de Reporte de Auditoría. En caso de que el Líder del proyecto indica desacuerdo con las recomendaciones registradas en el proceso de reporte de auditoría, la disposición final de recomendaciones del informe se hace por el Gerente del proyecto.

REPORTE DEL PROCESO DE AUDITORIAS

IDENTIFICADOR DE SEGUIMIENTO:_____

AUDITOR:_____FECHA DE REPORTE:_____

EQUIPO:_____

_____

NOMBRE DEL
PROYECYO:_____

FECHA DE LA AUDITORIA:_____

PROCESO/PROCEDIMIENTO
AUDITADO:_____

CHECKLIST DE AUDITORIA USADO
(Agregar)_____

RESULTADO DE LA AUDITORIA: (Marcar uno solo.)

_____ Proceso/Procedimiento Aceptable

_____ Proceso/Procedimiento Condicionalmente aceptable
(Sujeto a la finalización satisfactoria de los puntos mencionados arriba)
Notas de Condiciones:

_____ Proceso/Procedimiento Inaceptable
(Sujeto a la finalización satisfactoria de los puntos mencionados arriba)
Notas de Condiciones:

_____

TITULO ENCARGADO: FECHA VENCIMIENTO:

_____

_____

_____

ACCIONES CORRECTIVAS:

_____

____

DISPOSICON: APROBADO CANCELADO APLAZADO

Administrador del proyecto: FECHA:

_____

Firma del encargado SQA: FECHA:

Figura 7-1. Reporte del proceso de Auditorias

7.1.2 Procedimiento de Escalamiento para la Resolución de no concurrencia en el Proceso de reporte de Auditoría

Al encontrase un problema de calidad en algún elemento de trabajo ya sea documento, código o producto de software se tendrá que realizar lo siguiente:

1. Primero se tratará con el creador de ese elemento,

a. Si existen problemas de desacuerdos en la resolución de del problema, el personal de calidad tendrá que notificar al Líder del proyecto para que este tome cartas en el asunto y de una solución al problema.

b. Si el Líder del proyecto no da una solución, se escalará el problema al Gerente el proyecto y él tomará la decisión final.

El personal de calidad conservará el registro original de las conclusiones tomadas y la resolución posterior en sus expedientes de auditoría.

EVALUACION DE HERRAMIENTAS DE SOFTWARE

SQA:_____ FECHA DE EVALUACION:_____

Herramienta de Software Evaluada:

Métodos o criterios utilizados en esta evaluación:

Resultados de la evaluación:

Acciones correctivas recomendadas

Acciones correctivas tomadas

Figura 7-2. Evaluación de herramientas de software

SECCION 8. HERRAMIENTAS, TECNICAS Y METODOLOGIA.

// Párrafo introductorio.

8.1 HERRAMIENTAS

Utilidades del sistema operativo, depuradores, documentos de ayuda, checklist, analizadores de estructuras, analizadores de código, auditorias de estándares, monitoreo de rendimiento, software de desarrollo, matrices de seguimiento de software, pruebas de generadores de casos.

Lenguajes: PHP, HTML5, CSS, JavaScript, XML, Java, J2EE.

Herramientas de diagramas UML: Astah.

Herramientas de bases de datos: MYSQL, MongoDB.

Herramienta de procesamiento de texto: Microsoft Word, Notepad++

Herramientas de apoyo: Internet, AngularJS, Ant, Maven, Gradle.

Herramientas de desarrollo: NetBeans, Dreamweaver, IntelliJ IDEA, Eclipse, SWT, JFace, EMF, GEF, GMF.

Comentado [GPZ8]: Qué diablos es esto, no tiene definición

8.2 TÉCNICAS

En las técnicas se incluirá la revisión de uso de estándares, inspecciones de software, rastreo de requerimientos, verificación y validación de diseño y requerimientos, mediciones y evaluaciones de fiabilidad, análisis de lógica de negocio.

Estándares: Codificación de Lenguajes, UML, Diseño de BD Lógicas y Físicas

Programación en Pares.

Programación orientada a objetos y Programación orientada a eventos.

8.3 METODOLOGÍAS

Estas son un grupo de técnicas y herramientas. Estas metodologías se deben de documentar para completar la tarea o actividad y proporcionar una descripción del proceso que se va a usar

Paradigma de programación: Orientado a Objetos y Orientado a Eventos.

Metodología de desarrollo: PSP

SECCIÓN 9. CONTROL DE CÓDIGO FUENTE

Comentado [AA9]: Está mal el formato chavo.

Comentado [GPZ10R9]: corregido

El propósito de esta sección es definir los métodos e instalaciones usadas para mantener, almacenar, resguardar y documentar las versiones del código fuente identificado durante todas las fases del proceso de ciclo de vida software. Se utilizará la herramienta Git.

Para el correcto control del código fuente se tendrán que considerar los siguientes aspectos:

Las leyendas que se utilizarán para las ramas se ven en Tabla 9.1:

Instancia	**Rama**	**Descripción, Instrucciones, Notas**
Estable	Stable	Acepta la unión de las ramas "Trabajando" y "Revisión".
Trabajando	Master	Acepta la unión de las ramas “Características/Problemas" y "Revisión".
Características / Problemas	Topic-*	Siempre deriva de la rama "Trabajando".
Revisión	Hotfix-*	Siempre deriva de la rama "Estable".

Tabla 9.1 Tabla de definición de ramas

Ramas principales

En el proyecto software siempre deberán existir al menos dos ramas, la rama principal denominada “Master” que contendrá la última versión y la rama secundaria denominada “Stable” que contendrá la última versión que es estable, es decir que todas sus características han sido probadas y verificadas. La rama “Stable” no deberá ser manipulada hasta que el código sea verificado en la revisión técnica.

Ramas de soporte

En el proyecto las ramas de soporte son usadas como auxiliar en el desarrollo paralelo entre los desarrolladores, facilitar el rastreo de características y asistir en la reparación de defectos. Estas ramas deberán tener un tiempo de vida corto ya que se eliminarán eventualmente, el tiempo de vida dependerá de que tipo de rama sea y las características involucradas.

Los tipos de ramas que se usaran son:

- Rama de características
- Rama de bug
- Rama de revisión

a) Rama de características

Estas ramas son usadas cuando se desarrolla una nueva característica, el tiempo que tardará en lanzarse la característica dependerá de las características de esta. Sin importar cuando se finalice la rama, está siempre hará unión con la rama master.

- Debe derivar de master.
- Debe unirse con master.
- La convención del nombre será: feature-<#identificador de la característica dado en Jira>.

b. Rama de bug

La rama de bug difiere de las demás ramas semánticamente. Estas ramas solo deben ser creadas cuando exista un bug en un módulo que deba ser reparado y unido en la siguiente iteración, por esta razón la vida de una rama de bug no debe ser mayor al tiempo de la iteración. Sin importar cuando se finalice la rama, está siempre hará unión con la rama master

- Debe derivar de master
- Debe unirse con master
- La convención del nombre será: bug-<# número de bug identificado en el proceso de desarrollo>

c. Rama de revisión

Una rama de revisión proviene de la necesidad de actuar de inmediato mediante un estado no deseado de una modulo en desarrollo. Además, debido a la urgencia, no se requiere una revisión para ser integrado durante la siguiente iteración. Debido a estos requisitos, una rama de revisiones siempre debe derivar a partir de una rama estable. Esto por dos razones:

- El desarrollo de la rama master puede continuar mientras la revisión está siendo dirigida.
- Una rama estable representa lo que aún está en producción.

APÉNDICES

APÉNDICE A. PLANTILLAS PARA PRUEBAS

A1: REGISTRO DEL RESULTADO DE PRUEBAS

Por cada prueba ejecutada realizar una tabla que contenga las características de la prueba, así como el resultado de estas.

ID del test	Mismo ID que en la descripción del test	Comentario	Decisión
Descripción del test			OK, NOK, POK, NR, NC
Condiciones Iniciales			
Entradas			
Salidas			
Resultados esperados			
Procedimiento de la prueba			
Número del paso	Acciones realizadas	Resultados esperados	Resultado

Por cada NOK, se debe crear al menos un nuevo ticket de “bug”.

APÉNDICE B. TABLA DE REGISTRO DE TIEMPOS

Por cada actividad realizada por un desarrollador se actualizará el siguiente registro:

REGISTRO DE TIEMPOS						
Desarrollador:				**Proyecto:**		
Fecha	**Inicio**	**Fin**	**Interrupción**	**Tiempo**	**Actividad**	**Comentario**

APÉNDICE C. LISTAS DE VERIFICACIÓN

C1: LISTA DE VERIFICACIÓN PARA EL CÓDIGO

Esta lista se utiliza al finalizar la codificación de un módulo, con la finalidad de asegurarse de que el código producido cumple con características que lo hagan legible y fácil de entender a la vista de otros desarrolladores. Se marcará con una equis en la columna de "Faltante" si el ítem no fue incluido, en "Incorrecto" si fue incorrectamente utilizado, en "Extra" si el ítem fue descubierto pero no se identificó inicialmente, y en "Total" se colocará el número de ítems faltantes o extras.

Categoría del defecto	**Faltante**	**Incorrecto**	**Extra**	**Total**
¿Las sentencias switch contienen todos los casos necesarios?				
¿La anidación del código es muy profunda (más de tres niveles de indentación dentro de sentencias lógicas)?				
¿Hay lógica booleana negativa?				
¿Las sentencias if-then-else están construidas correctamente?				
¿Las sentencias do-while están construidas correctamente?				
¿Hay expresiones booleanas compuestas?				
¿Cada función se encarga de realizar una única cosa?				
¿Los nombres de las funciones y los métodos dicen qué es lo que hacen claramente?				
¿Los nombres de las variables dicen qué es lo que son claramente?				
¿Las constantes están escritas en mayúsculas?				