Testing en el Desarrollo de Software Científico en el Marco de la Integración Continua

Salamon Alicia,
Maller Patricio,
Mira Natalia,
Departamento de Informática
Instituto Universitario Aeronáutico
Av. Fuerza Aérea, 6500, Córdoba, Argentina

Boggio Alejandra,
Pérez Sofía,
Coenda Francisco.
Departamento de Informática
Instituto Universitario Aeronáutico
Av. Fuerza Aérea, 6500, Córdoba, Argentina

as.salamon@gmail.com, pmaller@gmail.com, ncmira@gmail.com, alejandra.boggio@gmail.com, sofiabeatrizperez@gmail.com, franciscocoenda@gmail.com

Abstract

El trabajo presenta una propuesta que provee a los desarrolladores del ámbito científico técnico, una forma de trabajo que incorpora las actividades de la etapa de pruebas dentro del ciclo de vida del desarrollo de software, en el marco de la integración continua, utilizando técnicas y patrones de pruebas basados en una norma de referencia específica del ámbito científico – técnico. De esta manera se intenta mejorar la calidad de los componentes desarrollados y formalizar la práctica de incluir el testing en el marco de la integración continua para adoptarla como referencia en futuros desarrollos. Este desarrollo está basado en el marco del proyecto de investigación PIDDEF 42/11, titulado "Metodología y Framework de Gestión de Líneas Base de Integración de Aplicabilidad en el Desarrollo de Software para el Proyecto UAV"

1. Introducción

La presencia de software crítico en nuestro entorno es cada vez mayor, lo encontramos encajeros automáticos, redes de comunicaciones, equipos médicos, aviones, automóviles, camiones, plantas de energía nuclear, etc., siendo necesaria su fiabilidad y seguridad para evitar muertes y/o perjuicios socio– económicos que pueden ocasionar a las personas.

Sin embargo, hasta en los sistemas más costosos, ampliamente probados y certificados pueden ocurrir fallos, y esto se debe a que estos sistemas son grandes y difíciles de comprender ya que contienen operaciones en tiempo real, algoritmos complejos, numerosas interacciones, etc.

Esta dificultad de probar la ausencia total de errores en los sistemas críticos seguirá vigente mientras no se modifiquen algunas prácticas, hay que fomentar la reutilización de los componentes de software ya probados en otros proyectos reduciendo así la presencia de fallos, como así también promover la publicación de los desarrollos de los sistemas críticos con el fin de compartir las experiencias y generar el aprendizaje colaborativo. Existen diferentes motivos por los cuales no se difunden los desarrollos de los proyectos, algunos de ellos son porque pertenecen al ámbito del Ministerio de Defensa y otros porque es una ventaja competitiva para la empresa que lo comercializa. Por ejemplo en el caso de la industria de la aviación, cada año se incrementa la presencia del software en los componentes del avión con lo que se hace difícil garantizar la ausencia total de errores [1].

No obstante, una de las respuestas de la ingeniería de software es la aplicación de un proceso de integración continua que aporta sistematización a todo el ciclo de vida del producto, contribuyendo en el descubrimiento de errores en etapas tempranas del proceso de desarrollo de software [2].

Por otro lado, los científicos y técnicos no cuentan con conocimientos profundos sobre la ingeniería de software, lo que conlleva a que el desarrollo de software científico–técnico carezca del soporte o fundamentos necesario para implementar las prácticas más básicas de esta disciplina. [3, 4].

Ante lo presentado anteriormente, se ha realizado una investigación en el marco del proyecto PIDDEF 42/11, llamado "Metodología y Framework de Gestión de Líneas Base de Integración de Aplicabilidad en el Desarrollo de Software para el Proyecto UAV", donde se

propone una arquitectura de referencia desarrollada con componentes open source [5] que permiten automatizar el proceso de desarrollo a través de la integración continua y el control de versiones.

Para un mejor relevamiento de la situación problemática en cuestión se llevó a cabo un Workshop con los equipos de desarrolladores de software científico–técnico de la institución, en virtud de su experiencia acumulada en este dominio. Dichos grupos trabajan a diario en el desarrollo de software crítico por lo que conocen cuales son los obstáculos a sortear para conseguir un producto de calidad. El objetivo del Workshop fue elicitar las complicaciones que se les presenta a los desarrolladores en el proceso de desarrollo de software científico–técnico.

Además, se relevó que estos equipos adoptan pautas de trabajo en sus proyectos a partir de la Norma European Cooperation For Space Standarization (ECCS). En la sección ECCS–E– ST-40C de dicha norma se trata la disciplina "Software" que se centra en los procesos de requisitos y en los resultados esperados, haciendo hincapié en la relación sistema-software y en la verificación y validación de elementos software. Esta sección se complementa con otras: ECSS–M–ST–40 (Configuración y gestión de la información), ECSS–Q–ST–20 (Gestión de los procesos) y ECSS–Q–ST–80 (Aseguramiento de la calidad) [6].

En respuesta a los resultados del Workshop, y a partir de los antecedentes investigados, se construyó como solución una arquitectura con el fin de mejorar la calidad del producto como también las buenas prácticas de desarrollo de software. En este trabajo se expone cómo aplicar algunos de los patrones y técnicas de testing de la ingeniería de software en el desarrollo de software crítico mediante la arquitectura de integración continua.

2. Software Científico

El desarrollo de software científico es muy diferente al desarrollo de software comercial, ya que poseen características particulares, hay dos culturas bien definidas [7]. Una de ellas es el dominio en el que se desenvuelven, tomando en cuenta que el ámbito científico es un ambiente muy específico, ya que sólo un grupo muy pequeño poseen un conocimiento acabado acerca de la problemática a solucionar. Desde hace un tiempo se ha tratado de transferir las metodologías y prácticas de la ingeniería de software al ámbito científico pero han sido un mal ajuste ya que no se ha tratado de entender a la comunidad científica y sus limitaciones [8,9].

Segal observó la existencia de dos culturas bien marcadas, donde los científicos saben del dominio acabadamente y escasamente poseen formación en cuanto a las metodologías de desarrollo de software, documentación formal, buenas prácticas de programación, gestión de equipos mientras que los ingenieros de software saben hacer software de calidad, flexible, reutilizable [10,11].

Los problemas antes expuestos aparecen ya que los desarrolladores científicos rara vez tienen formación acerca de la ingeniería de software [3,4].

3. Integración Continua

La integración continua es una práctica en la cual los miembros de un grupo de desarrollo integran (compilación y ejecución) los distintos componentes de un proyecto con una frecuencia especificada. Cada integración se realiza de forma automática (incluyendo sus test) con el fin de detectar los errores de integración lo antes posible. Según Martin Fowler, muchos equipos de desarrollo han encontrado que este enfoque reduce significativamente los problemas de integración y permite que los equipos desarrollen software cohesivo más rápido [2].

Un escenario típico de Integración Continua [12] se compone de la siguiente manera:

- Paso 1: Un desarrollador realiza un commit de su trabajo al repositorio de control de versión a la vez que el servidor de Integración Continua verifica cambios en el repositorio, por ejemplo cada 5 minutos.
- Paso 2: El servidor de integración continua detecta los cambios en el repositorio de control de versión, extrayendo el último commit que se ha realizado y ejecutando una build script que se encarga de integrar los distintos componentes del software en desarrollo.
- Paso 3: El servidor de integración continua genera feedback con los resultados del proceso de building, el cual es enviado a los miembros que se especifique del proyecto.
- Paso 4: Sólo si el paso anterior se ejecutó correctamente. El servidor de integración continua genera feedback con los resultados del proceso de testing, el cual es enviado a los miembros que se especifique del proyecto.
- Paso 5: El servidor de integración continúa revisando cambios en el repositorio de control de versiones.

La Fig. 1. muestra las partes de este escenario que se ha descripto.

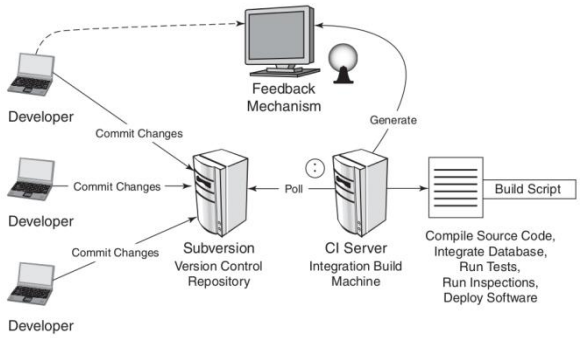

Fig. 1. Escenario típico de integración continua

4. Pruebas de Software o Testing

Como antes se mencionó las pruebas de software son uno de los módulos de la arquitectura de la integración continua que se encuentra dentro en el repositorio y es ejecutado por el servidor de integración continua luego de que el software se construyó correctamente. [2]

Según Cem Kaner [13] define el testing como una investigación técnica de un producto bajo prueba con el fin de brindar información relativa a la calidad del software. Es decir, dar cierto nivel de confianza a los interesados. Para ello Martin Fowler [2] recomienda la automatización de las pruebas como una buena práctica de la integración continua ya que permite capturar muchos bugs a lo largo del desarrollo del producto, pero la excesiva cantidad de ejecución de pruebas hace que la integración continua sea lenta entonces Paul Duvall agrega que es necesario una categorización para las pruebas con el fin de que el sistema de integración continua no demore cada vez más en completar las construcciones a medida que el proyecto avanza [12].

Por otra parte Myers [14] afirma que las pruebas de software requieren más creatividad que su diseño y se debe a que es una tarea extremadamente creativa e intelectualmente desafiante, de hecho muchos ingenieros que diseñan y desarrollan software admiten que encuentran mayor satisfacción en el diseño de las pruebas ya que implica un desafío encontrar errores; es por ello que algunas personas lo llaman "quebrar el software". Pero esta acepción es errónea ya que los ingenieros de prueba realmente no rompen software sino simplemente exponen los errores que ya existen, al refutar o falsear el supuesto de que el software es impecable a través de los resultados que observan, luego de aplicar pruebas estructuradas, de deducción lógica y de experimentos.

Sin embargo, los errores que se encuentran sólo son una parte de las pruebas ya que los ingenieros de pruebas además agregan valor al proceso de desarrollo mediante la validación del software y la detección de errores, facilitándoles a los desarrolladores información para evitar errores y a la administración para evaluar el riesgo.

Para cumplir correctamente el labor de los ingenieros de pruebas de software es necesario poseer ciertas capacidades intelectuales y personales, James McCaffrey define ocho cualidades: [15]

- Pasión por el análisis y las pruebas
- Habilidades técnicas
- Capacidad Intelectual pura
- Capacidad para priorizar y organizar
- Capacidad de adaptación y aprendizaje
- Capacidad para trabajar sin supervisión directa
- Capacidad para comunicarse
- Capacidad para comprender la estrategia de negocios

Como los sistemas de software crítico aumentan su tamaño y complejidad a medida que se van desarrollando, el valor de las pruebas de software se está incrementando y convirtiendo en una actividad cada vez más importante.

Por otra parte, para el desarrollo de sistemas críticos las normas ECSS, por ejemplo, define métodos y técnicas de testeo basándose en diversos patrones de testing.

Sin embargo, dentro de los ámbitos de desarrollo de software científico–técnico no se visualiza una amplia utilización de pruebas, a pesar de que muchos desarrolladores de estos sistemas reconocen el valor e importancia de estos en el desarrollo de sistemas [16]. También se sigue visualizando una problemática en la cual los equipos de desarrollo y en especial los de testing no se encuentran envuelto desde el inicio del proyecto, es decir, desde la definición de los requerimientos. Esto ha podido ser corroborado en las observaciones que se realizaron en grupos de I+D donde la inclusión, en especial del equipo de testing, suele ser tardía [17].

5. Proceso de prueba

Un proceso de pruebas formal, está compuesto por cinco etapas [18].

1. Planeación de pruebas.
2. Diseño de pruebas.
3. Implementación de pruebas.
4. Evaluación de criterios de salida.
5. Cierre del proceso.

A continuación se presenta en la figura 2 el proceso de pruebas identificando sus entregables de entradas y salidas.

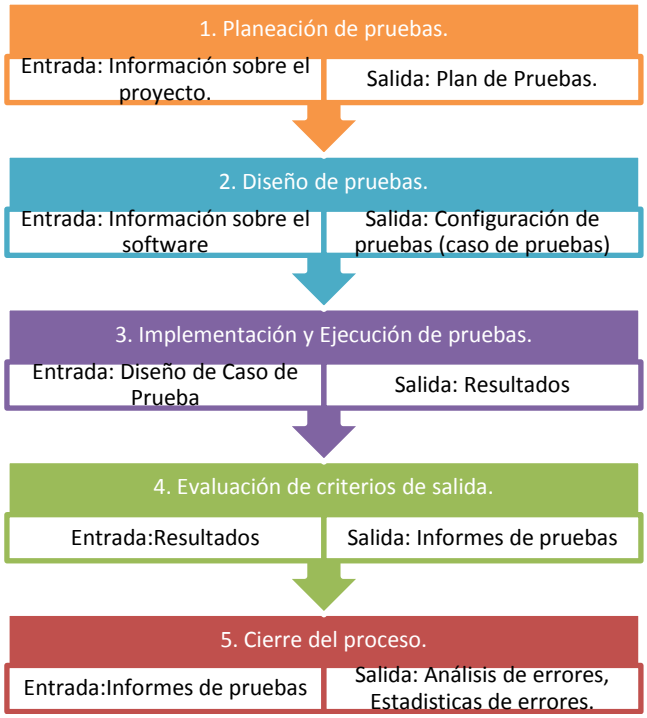

Fig. 2. Proceso de prueba.

1. Planeación de Pruebas: se genera el artefacto o entregable denominado "Plan de Pruebas" el cual debe estar conformado mínimamente por los siguientes aspectos:
    - Alcance de la prueba: en esta etapa se determina que funcionalidades del sistema serán probadas durante el transcurso de la prueba como también el alcance detallado del proceso de pruebas, la prioridad con la que las funcionalidades deben probarse y la profundidad de las pruebas. Este listado de funcionalidades a probar se extrae del análisis de riesgos realizado en una etapa previa donde se tienen en cuenta variables tales como el impacto que podría ocasionar la falla de una funcionalidad y la probabilidad de falla de una funcionalidad.
    - Tipos de Prueba: se define qué tipos de pruebas requeriría el producto. No todos los productos de software requieren la aplicación de todos los tipos de pruebas que existen, es por ello que el líder de pruebas deberá plantearse qué tipos de prueba son aplicables al proyecto en evaluación.
    - Estrategia de Pruebas: como no es viable probar todas las posibles combinaciones de datos, es necesario establecer a través de un análisis de riesgos sobre que funcionalidades se debe centrar mayor atención. Adicionalmente, una buena estrategia de pruebas debe indicar los niveles de pruebas o ciclos que deberán emplearse y la profundidad a aplicar para cada nivel de pruebas definido como también definir los criterios de entrada y salida para cada ciclo de pruebas a ejecutar.
    - Criterios de Salida: se debe especificar de manera formal, bajo qué condiciones se puede considerar que una actividad de pruebas fue finalizada. Los criterios de salida se deben definir para cada nivel de pruebas a ejecutar. Por ejemplo el porcentaje de funcionalidades de alto riesgo probadas con éxito, número defectos críticos y/o mayores aceptados, etc.
    - Otros aspectos: se debe incluir una estimación de tiempos, los roles y/o recursos que harán parte del proceso, la preparación del entorno de pruebas, cronograma, etc.
2. Diseño de Pruebas: una vez elaborado y aprobado el "Plan de Pruebas", el equipo de trabajo debe iniciar el análisis de toda la documentación existente con respecto al sistema para diseñar los casos de prueba. Los entregables claves a revisar son: casos de uso, historias de usuario, arquitectura del sistema, diseños, manuales de usuario, manuales técnicos, etc. El diseño de los casos de prueba, debe considerar la elaboración de casos positivos y negativos. Estos últimos permiten validar cómo se comporta el sistema ante situaciones atípicas y permite verificar la robustez del sistema, atributo que constituye uno de los requerimientos no funcionales indispensable para cualquier software crítico. Por último, es necesario definir cuáles son los datos de prueba necesarios para la ejecución de los casos de prueba diseñados.
3. Implementación y Ejecución de Pruebas: en esta etapa se inicia con la creación de los datos de prueba necesarios para ejecutar los casos de prueba diseñados en la etapa 2. La ejecución de estos casos, puede realizarse de manera manual o automatizada; en cualquiera de los casos, cuando se detecte un fallo en el software, este debe ser documentado y registrado en una herramienta que permita gestionar los defectos. Una vez el defecto ha sido corregido, es necesario realizar un re–test que permita confirmar que el defecto fue solucionado de manera exitosa. Por último, es

indispensable ejecutar un ciclo de regresión que permita garantizar, que los defectos corregidos en el proceso de depuración no hayan ocasionado más defectos en el software.

4. Evaluación de Criterios de Salida: los mismos son necesarios para determinar si es posible dar por terminado un ciclo de pruebas. Por lo que es conveniente definir una serie de indicadores que permitirán comparar los resultados obtenidos contra los indicadores definidos, sí los resultados obtenidos no superan los indicadores definidos, no es posible continuar con el siguiente ciclo de pruebas. Coexisten varios tipos de criterios de salida por ejemplo cubrimiento de funcionalidades en general, cubrimiento de funcionalidades críticas para el sistema, número de defectos críticos y mayores detectados, etc.
5. Cierre del proceso: principalmente en esta etapa se elabora un informe con del análisis de errores encontrados a lo largo del proceso de prueba como también una estadística de los errores más frecuentes dejando lecciones aprendidas para aplicar en futuros proyectos. Además, se comprueba que se hayan cerrado todas las incidencias reportadas, se verifica que los artefactos de software hayan sido entregados y aprobados, etc.

5.1. Técnicas de pruebas de software crítico

Según la Norma ECSS en el documento ECSS–E–HB–40A [6] y SWEBOK [19], describen las siguientes técnicas de testeo.

- Introducción de fallos: consiste en ingresar fallos en el hardware y software de manera intencional con el fin de probar los mecanismos de tolerancia de fallos, para que la misma sea eficaz es necesario contar con una cantidad de tiempo significativa de un analista que posea una buena perspectiva del software y con herramientas automatizadas así como lo propone la integración continua. Esta técnica apunta a la evaluación de la robustez del sistema.
- Prueba de estrés: en este caso se busca sobrecargar el sistema de manera que active sus mecanismos de manejo de errores y recuperación de sí mismo. Por ejemplo son objetos a controlar los buffers internos o variables dinámicas, pilas como también el uso de los recursos del sistema, tiempo CPU, espacio de almacenamiento y memoria. Esta técnica hace hincapié en probar el rendimiento del sistema.
- Partición equivalente y clases equivalentes: se divide el dominio de entrada de un software en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para la partición equivalente se basa en la evaluación de las clases de equivalencia. Normalmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. Los mismos criterios se aplican a las salidas esperadas. El objetivo de esta técnica es generar resultados en todas y cada una de las clases, de esta manera se reduce el número de casos de pruebas.
- Análisis de valores límites: el propósito es proporcionar casos de prueba para detectar y eliminar los fallos que se producen en los límites de los parámetros para ello es necesario identificar las clases equivalentes. Esta técnica se emplea en conjunto con la técnica antes mencionada en virtud de garantizar la cobertura de las pruebas especificadas. El análisis de valores límites permite detectar fácilmente las fallas de cálculo, tamaño de la matriz, punteros nulos, iteraciones de bucle.
- Previsión de Errores: este tipo de test es diseñado por ingenieros de software cuyo objetivo es anticiparse a las posibles fallas del sistema. Para ello, se utiliza la información histórica de aquellas fallas que se descubrieron en proyectos anteriores.
- Tabla de decisión: la tabla de decisión representa lógicamente los imput y output. Los casos de test se derivan considerando cada combinación posible entre input y output.
- Observación del Usuario: la especialización heurística, también conocida como inspección de usabilidad, es aplicada de manera sistemática a la observación del uso de sistemas en condiciones controladas con la finalidad de determinar cómo los usuarios finales utilizan el sistema y sus interfaces.

5.2. Patrones de Testing

La IEEE en su libro SWEBOK define una serie de patrones de testing [19]. De estos patrones de testing que se proponen a continuación se toman solamente algunos para ser incluidos en la arquitectura propuesta de este trabajo.

- Test Unitario: Consiste en la verificación de manera aislada de partes del software que son factibles de ser testeada. Estas pruebas son atómicas y son escrita normalmente por los mismos desarrolladores.
- Test de Integración: Se ocupa de verificar la interacción entre los componentes de software. En la actualidad una práctica que está tomando mayor auge es la integración incremental de testing, lo que permite ir verificando los resultados de integración a través de cada iteración sin necesidad de hacer una gran integración al final del desarrollo.

- Test de Regresión: Es una re–testeo selectivo de los componentes de un sistema con el fin de verificar que las modificaciones introducidas no produzcan efectos no deseados y que los componentes sigan cumpliendo con los requerimientos para los que fueron desarrollados.
- Test de estrés: En este tipo de test evalúa la carga máxima y minina de trabajo que soporta el sistema y los mecanismos de protección que poseen los sistemas críticos.
- Test de Recuperación: En este tipo de test se evalúa las capacidades del sistema de reiniciarse después de que se produce una falla o algún otro tipo de contingencia.
- Test de Sistema: Este test evalúa el comportamiento del sistema en general. Suele usarse para evaluar los requerimientos no funcionales, como seguridad, velocidad, confiabilidad, etc.
- Test de aceptación: Se evalúa si el sistema los criterios de aceptación a través del chequeo del comportamiento del sistema contra los requerimientos del cliente.
- Test de seguridad: Se verifica que el sistema se encuentre protegido ante ataques externos. Se verifica la confidencialidad, integridad y disponibilidad del sistema y la información.
- Test de Humo: Consiste en realizar pruebas rápidas con el fin de testear que la funcionalidad básica del sistema se encuentra estable.

6. Propuesta

La propuesta de este trabajo es elaborar una solución que incorpore las actividades de la etapa de pruebas del software al proceso de desarrollo del software científico técnico, sugiriendo algunas técnicas y patrones de testing en el marco de trabajo del PIDDEF 42/11. Cabe destacar que los patrones recomendados están contemplados en la norma ESCC y SWEBOK.

6.1. Técnicas y patrones sugeridos

De acuerdo a las características mencionadas de los grupos de desarrollo científico–técnico, como ser la escasa formación en cuanto a la ingeniería de software se ha optado por seleccionar técnicas sencillas de aplicar y que permitan vislumbrar resultados efectivos rápidamente.

Las técnicas que se proponen utilizar son: partición equivalente y clases equivalentes, y análisis de valores límites. Además, se seleccionan los patrones de pruebas de unidad, de integración, regresión, y de humo. Las técnicas y patrones seleccionados deben ser aplicados siguiendo la forma de trabajo que se describe a continuación.

6.2. Forma de trabajo

Se adopta el proceso planteado para la etapa de pruebas:

- Planificar las pruebas a realizar: se debe generar el Plan de Pruebas, en el mismo se especifican los siguientes aspectos: alcance de cada prueba, tipos de pruebas, estrategias de pruebas, criterios de salida, estimación de tiempo de las pruebas, los roles y recursos involucrados en cada una de las pruebas, cronograma, etc.
- Pre–definir los resultados esperados: en esta actividad se utilizan las técnicas de partición equivalente y clases equivalentes y análisis de valores límites y los patrones asociados.
- Construir las pruebas: se construyen los componentes de pruebas para ser ejecutadas en el marco de la integración continua. Cabe aclarar que los componentes de prueba una vez construidos deben ser ubicados en un lugar específico de la estructura de directorio del servidor de integración continua.
- Ejecutar las pruebas: la configuración del servidor de integración continua genera la ejecución del código y de los componentes de prueba asociados al mismo.
- Evaluar las pruebas: determinar si el componente de prueba es adecuado para la validación del código.

6.3. Equipo involucrado

Es importante determinar quien o quienes del equipo de desarrollo poseen un perfil más cercano al descripto por McCaffrey. Se sugiere buscar personas que sean creativos para diseñar las pruebas, que logren el objetivo de validar software encontrando errores y proporcionando a los desarrolladores información para evitar errores futuros [15].

Asimismo existen diferentes roles que participan en un proyecto de pruebas. Una persona puede asumir más de un rol en un proyecto y tomar diferentes roles en proyectos distintos. En la medida de que una persona es capaz de asumir distintos roles va creciendo en conocimiento y experiencia.

Se identifican tres tipos de roles dentro de un equipo de testing: tester de software, líder de testing de software y consultor.

Tester de software es quien diseña, ejecuta e informa el resultado de las pruebas sobre un producto de software. Los testers integran el equipo de pruebas e interactúan con el equipo de desarrollo y la gerencia de proyectos.

Líder de testing de software es quien está a cargo del equipo de testing y tiene conocimiento y experiencia en

el diseño, ejecución y reporte de pruebas sobre un producto de software, pero también es capaz de estimar, planificar y dar seguimiento a un proyecto de pruebas.

Consultor. Es un rol adecuado para aquellas personas capaces de dirigir un equipo de pruebas y orientar sobre mejores prácticas en testing [16].

En el ámbito científico en la mayoría de los casos quien realiza las pruebas es la misma persona que desarrolla el componente a ser probado, pero no es conveniente ya que la persona conoce como está programado, es por ello que es recomendable que los miembros del equipo asuma uno o más roles diferente y logre cumplir con los objetivos de las pruebas.

6.4. Las pruebas en el marco de la integración continúa

La arquitectura de integración continua del PIDDEF 42/11 se compone de un servidor de integración continua (Cruise Control), un sistema de control de versiones (Git) y un sistema de construcción (Ant).

El proceso da inicio con la creación de un repositorio vació donde se va a jerarquizar el desarrollo de software. Para esto se arma una estructura de directorios donde se separa el código fuente de la suite de test y a su vez, esta última se la jerarquiza en componente por componente de a acuerdo a la funcionalidad que se pretende validar y verificar [12]. Los desarrolladores remiten sus modificaciones al repositorio del proyecto, el cual alimenta al servidor de integración continua en su proceso de construcción. Este último presenta los resultados del proceso de construcción a través de su interfaz web.

En este proceso de construcción, se recomienda que los test unitarios sean ejecutados con cada construcción [20], mientras que aquellos componentes que han finalizado sus test no es necesario que los corran nuevamente. En cuanto a los test de integración y de humo se sugiere que sean ejecutados una vez que se ha finalizado con el desarrollo del componente. Estas pruebas deberán ser planificadas y en la medida de lo posible ser ejecutadas fuera del horario laboral, por ejemplo a la noche [20]. Por último, los test de regresión deberían ser ejecutados cuando se produzca un fallo en las pruebas de humo e integración. Este tipo de test se recomienda que sea ejecutado los fines de semana.

Los test unitarios van a permitir evaluar los input–output, permitiéndole a los desarrolladores de software científico–técnico verificar y validar sus modelos [20]. Los test de integración y humo van a permitir corroborar la interacción de los nuevos módulos que se agreguen al sistema y verificar que se siga manteniendo la estabilidad de este [19,20]. Las pruebas de regresión van a permitir determinar que las modificaciones introducidas en el sistema no produzcan fallas no deseadas en otras partes del sistema y que este siga cumpliendo con los requerimientos iniciales [21]. A través de esta planificación se busca evitar que los tiempos de integración se incrementen exponencialmente a medida que el desarrollo avanza.

7. Resultado

El modelo propuesto se ha presentado al grupo de desarrollo que ha participado en el Workshop logrando captar su interés y una amplio involucramiento, en la exposición se vieron comprometidos y con deseos de experimentar la implementación de esta solución, incorporando a su forma de trabajo el proceso formal de pruebas descripto con sus técnicas y patrones. El próximo paso es realizar la experiencia evaluando los indicadores definidos para tal fin.

8. Conclusión

La investigación y desarrollo que se ha llevado a cabo en este trabajo ha permitido obtener un conocimiento mayor acerca de la problemática que viven los equipos de desarrolladores de software científico y especialmente de aquellos grupos que desarrollan sistemas críticos en el cual no se permiten fallas ya que las consecuencias implican el riesgo de vidas humanas y una considerable suma de dinero.

Por lo tanto, a través de este trabajo se busca plantear un modelo de referencia que permita mejorar el desarrollo de componentes de software crítico en el marco de la integración continua donde se busca integrar y testear automáticamente el desarrollo de un proyecto a medida que este avanza. De esta manera se posibilita el descubrimiento de errores a corto plazo sin tener que esperar a realizar las pruebas finales. También se espera aportar una reducción del re trabajo y la optimización de los tiempos de desarrollo.

Este trabajo continuará con la realización de una experiencia con el grupo de desarrollo de software científico – técnico para la aplicación de esta nueva forma de trabajo a través de la arquitectura desarrollada en el proyecto PIDDEF 42/11.

9. Referencias

[1] Sánchez Domínguez J. C. y Rodríguez Dapena P. "¿Cómo verificar la calidad del software en sistemas críticos?", *Ponencia del IV Congreso Gallego de la Calidad*, Santiago de Compostela, España (2003).

[2] Continuous Integration por Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html

[3] D. Kelly y R. Sanders. "Assessing the Quality of Scientific Software". *First International Workshop on Software Engineering for Computational Science and*

Engineering. Leipzig, Germany. (2008).
[4] Carver J., Kendall R., Squires S., y Post D. "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies."*IEEE Computer Society*, Washington, DC, USA, (2007).
[5] The Open Source Initiative, http://opensource.org/
[6] Normas ECSS http://www.ecss.nl/
[7] Segal J. "Scientists and software engineers: a tale of two cultures". *PPIG University of Lancaster*, UK, (2008).
[8] Kelly D. "A Software Chasm: Software Engineering and Scientific Computing."*IEEE Computer Society.* Los Alamitos, CA, USA. (2007).
[9] Segal J. "Models of scientific software development." *First International Workshop on Software Engineering in Computational Science and Engineering*, Leipzig, Germany. (2008).
[10] Basili V. et al. "Understading the High-Performance-Computing Community: A Software Engineer's Perspective." *IEEE Software Computer Society.* vol. 25, no. 4, 29-36. Los Alamitos, CA, USA. (2008).
[11] Hannay J. E., MacLeod C., y Singer J. "How Do Scientists Develop and Use Scientific Software?" *IEEE Computer Society*, Washington, DC, USA, (2009).
[12] Duvall P., Matyas S. y Glover Andrew. *Continuous Integration: Improving Software Quality and Reducing Risk*. (2007).
[13] Kaner C., Falk J. y Nguyen H. *"Testing Computer Software"* International Thomson Computer Press. (1993)
[14] Myers, G. *"The Art of Software Testing".* John Wiley & Sons. 2004.
[15] McCaffrey J. *"What Makes A Good Software Tester?"* 2008.
[16] Nguyec-Hoan L., Flint S. y Sankaranarayana R.*"A Survey of Scientific Software Development".* ESEM, Bolzano-Bozen, Italia(2010).
[17] Humphrey W., Snyder T. y Willis R. "Software Process Improvement at Hughes Aircraft". *IEEE Software Computer Society Vol. 8 Issue 4* (1991).
[18] Jacobson I., Booch G. y Rumbaugh.J. *"The Unified Software Development Process."* Addison-Wesley Professional. 1999.
[19] Bourque P. y Fairly R. *"SWEBOK v 3,0 Guide to the Software Engieering Body of Knowledge."* IEEE. Accedido el 2014.
[20] Stephen P. B., Appleton B. y Brown K. *"Software Configuration Management Patterns: Effective Teamwork, Practical Integration Paperback"*. Addison Wesley 2002.
[21] Cardinal, M. *"The Hidden Roles of Software Architects".* MSDN. 2008.