Methodology to teach to assure the software quality through verification and validation techniques.

Metodología para enseñar a asegurar la calidad del software a través de técnicas de verificación y validación.

Gabriela Salazar Bermúdez
Escuela de Ciencias de Computación e Informática, Universidad de Costa Rica,
San José Costa Rica.
gabriela.salazar@ecci.ucr.ac.cr

ABSTRACT
The main objective of this article is to describe the experience of teaching the students of Software Engineering of the Bachelor Program in Informatics of the University of Costa Rica, to apply validation and verification software tests. The purpose is to extend the software quality assurance culture, facilitating and stimulating the use of worldwide methodologies in their future development as professionals; in order to support the national companies to improve their competitivity in international markets. Specifically, it describes the procedure in the execution of these tests, forms used, the software tools and the team members of the process. The points described in this article can be of interest for teachers and instructors who wish to develop software engineers in the area of software quality.

Resumen
El objetivo principal de este artículo es describir la experiencia de enseñar a los estudiantes del curso de Ingeniería de Software del Programa de Bachillerato en Computación e Informática en la Universidad de Costa Rica, a aplicar pruebas de verificación y validación del software. El propósito es extender la cultura de aseguramiento de calidad del software, facilitando y estimulando el uso de metodologías de alcance mundial en los futuros profesionales, con el fin de apoyar a las empresas nacionales a mejorar la competitividad en mercados internacionales. Específicamente se describe el procedimiento en la ejecución de estas pruebas, los formularios utilizados, las herramientas de software y los participantes del proceso. Los puntos descritos en este artículo pueden interesar a profesores e instructores que deseen formar a futuros ingenieros de software en el campo de la calidad del software.

Categories and Subject Descriptors
D.2 [**Software Engineering**] D.2.4 [**Software/Program Verification Language**]: D.2.5 [**Testing and Debugging**]

General Terms
Standardization, Verification, Documentation, Experimentation.

Keywords
Software Quality Assurance, Verification, Validation, Formal Technical Review, Testing, Standards, Software Engineering Education.

Palabras clave
Aseguramiento de Calidad del Software, Verificación, Validación, Revisión Técnica Formal, Pruebas, Estándares, Enseñanza de Ingeniería de Software.

1. INTRODUCCIÓN
El aseguramiento de la calidad es en la actualidad uno de los tópicos de investigación más importantes dentro del área de ingeniería de software. La falta de calidad de los sistemas que se desarrollan es uno de los mayores contribuyentes a la llamada “crisis” del software actual. [21]

A nivel mundial existen múltiples metodologías, técnicas y herramientas modernas de ingeniería de software que permiten mejorar la calidad de los productos de software desarrollados. Algunas de ellas son: el Programa de Certificación de Pruebas de Software (Certified Software Tester, en sus siglas en inglés CSTE) [19], los estándares de ingeniería de software del IEEE [10], el Modelo de Capacidad/Madurez (CMM) [16-20] y el estándar ISO 9001 [11].

Las instituciones y las empresas nacionales que desarrollan o adquieren software requieren un apoyo importante en el proceso de introducir metodologías de aseguramiento de la calidad del software. El estimular a los estudiantes en el uso de metodologías de alcance mundial, les facilita su entrada al mercado laboral y se convierte en un apoyo importante para la industria desarrolladora de software del país.

En la próxima sección se describe el marco teórico de las metodologías que fueron aplicadas. En la sección tres se describe brevemente el curso de Ingeniería de Software utilizado en esta experiencia. En la sección cuatro se explica cómo se aplicaron dichas metodologías y finalmente, en la sección cinco se describen las conclusiones.

2. MARCO TEÓRICO
2.1 Aseguramiento de la Calidad del Software
La ingeniería de software es la disciplina que desarrolla y utiliza metodologías, métodos y herramientas para desarrollar software de buena calidad. Uno de los componentes claves de todo proceso de desarrollo de software, son las actividades que se llevan a cabo para asegurar la calidad del software que se produce. Este conjunto planeado y sistemático de actividades que aseguran que el proceso y

los productos cumplen con los requerimientos técnicos establecidos, se conoce como aseguramiento de la calidad del software (ACS). Un software de calidad es aquel que: "...concuerde los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas (por ej.: fácil de mantener) que se esperan de todo software desarrollado profesionalmente". [14-19]

De acuerdo a [7] un componente importante del ACS son las actividades de verificación y validación (V&V) del software que se realizan durante las diferentes fases que componen el ciclo de desarrollo de los sistemas. El proceso de V&V provee una evaluación objetiva de los productos y del proceso desarrollados durante el ciclo de vida del software. Esta evaluación demuestra que los requerimientos del software y del sistema son correctos, completos, precisos, consistentes y fáciles de probar. Otros objetivos de V&V son:

a. Facilitar la detección y corrección temprana de errores,

b. mejorar la administración del proceso y los riesgos del producto, y

c. apoyar el proceso del ciclo de vida para asegurar el cumplimiento de los requerimientos de rendimiento, cronograma y presupuesto del programa.

El proceso de V&V provee evidencia de que el software y los productos asociados:

a. Cumplen con los requerimientos en todas las actividades durante el proceso de desarrollo.

b. Satisfacen los estándares, prácticas y convenciones durante el proceso de desarrollo.

c. Establecen una base para evaluar cada actividad del ciclo de vida para iniciar la siguiente actividad.

Una técnica importante de V&V son las revisiones técnicas formales, las cuales ayudan a verificar y validar los productos de software, que se desarrollan durante las diferentes fases que componen el ciclo de vida del proyecto.

El verificar la calidad en cada una de las fases de desarrollo y no esperarnos hasta la fase de pruebas, reduce el esfuerzo y la duración del ciclo de vida, ya que lo que se persigue es disminuir los costos de prueba y de integración y un menor número de cambios en las primeras versiones del producto de software. Además, el proceso de revisiones como parte de las actividades de aseguramiento de calidad, hace que se detecten y se eliminen un gran porcentaje de defectos (en organizaciones con experiencia del 60% al 90%). [17-19]

2.2 Pruebas de software

De acuerdo a [4] las pruebas de software son un conjunto de actividades diseñadas para evaluar la calidad de los productos desarrollados. Por su lado Myers en [14] define las pruebas del software como las actividades de planificar, diseñar, desarrollar, administrar y ejecutar las pruebas, incluyendo también actividades de verificación y validación.

A continuación se describen brevemente las actividades que deben realizarse en un proceso de pruebas.

Planificación: Con base en la especificación funcional y la información del proyecto se elabora un plan de pruebas que incluye: objetivos, estrategias, recursos, administración de los riesgos y el cronograma. Algunos aspectos que se consideran dentro de la estrategia son: la fase de prueba (unitaria, de integración, etc.), el tipo (funcional, de rendimiento, etc.), la técnica (caja blanca, caja negra), las herramientas y los criterios de completitud.

Diseño: Se refina la estrategia de pruebas definida en el plan, se definen procedimientos de pruebas, se especifican los casos de prueba y los criterios de aceptación. Los procedimientos se especifican con sus correspondientes referencias a scripts (si fuera necesario) y a los casos de pruebas ya definidos.

Desarrollo: Se crean o adquieren pruebas automatizadas y se prepara el ambiente de pruebas de acuerdo a las condiciones planificadas en cuanto a: hardware, software, espacio de laboratorio y de recursos humanos. Lo ideal es automatizar las pruebas para poderlas reutilizar y mantener un rastreo de las pruebas y validación de requerimientos.

Ejecución: Se ejecutan las pruebas y se registran los hallazgos. Posteriormente se prepara una bitácora de pruebas y un reporte de defectos.

Evaluación de resultados: Se revisan los reportes y se generan un resumen de métricas, de cobertura de pruebas y de análisis de defectos.

2.3 Niveles de pruebas [19]

El ciclo de vida del proceso de pruebas involucra pruebas continuas del sistema que se realizan en paralelo con el proceso de desarrollo. Por ejemplo mientras el equipo de desarrollo especifica los requerimientos el de pruebas planifica las pruebas con base en los requerimientos. Durante el proceso de desarrollo en puntos predeterminados, los resultados son inspeccionados para determinar su correctitud. Refiérase a la Figura 1.

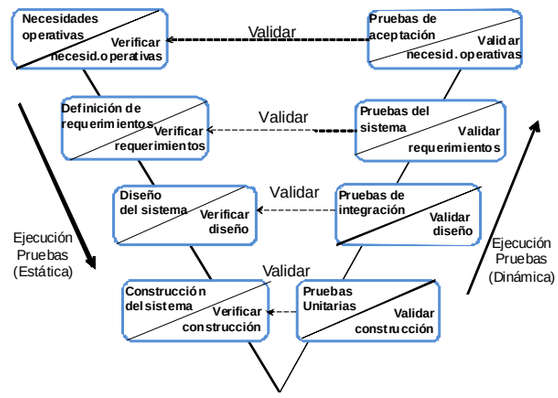

Figura 1. El concepto "V" de pruebas de software

En la Figura 1 se puede observar también que se aplican pruebas estáticas sobre los entregables desde el inicio del desarrollo, y una vez que el código se encuentra en un estado ejecutable, se realizan pruebas dinámicas para verificar que los requerimientos especificados se están alcanzando.

La secuencia en que ocurren las pruebas es representada por los siguientes tipos de pruebas: 1. Pruebas de verificación. 2. Pruebas unitarias. 3 Pruebas de integración. 4 Pruebas del sistema. 5. Pruebas de aceptación. A continuación se describe cada tipo.

Pruebas de verificación: Son pruebas estáticas sobre entregables intermedios, que se producen durante el ciclo de vida del proyecto. Aseguran que el sistema (software, hardware, documentación y personal) cumpla con los estándares y procesos de la organización. Utilizan técnicas como: revisiones, caminatas o inspecciones, que no requieren ejecutar código.

Pruebas unitarias: Consisten en probar piezas de software pequeñas; a nivel de secciones, procedimientos, funciones y módulos. Dichas pruebas se utilizan para asegurar el correcto funcionamiento de secciones de código, mucho más reducidas que el conjunto, y que tienen funciones concretas con cierto grado de independencia. Utilizan técnicas que ejercitan rutas específicas en una estructura de control de componentes, para asegurar una cobertura completa y la máxima detección de errores.

Pruebas de integración: Se realizan una vez que las pruebas unitarias fueron concluidas exitosamente; con éstas se intenta asegurar que el sistema completo, incluso los subsistemas que componen las piezas individuales grandes del software funcionen correctamente al operar e interoperar en conjunto. Se utilizan las técnicas de diseño de casos de pruebas que se enfocan en entradas y salidas, aunque también pueden usarse técnicas que ejerciten rutas de programas específicas para asegurar la cobertura de las principales rutas de control.

Pruebas de regresión: La prueba de regresión se enfoca en una nueva ejecución de algún subconjunto de pruebas que ya se ha realizado, con el fin de asegurar que los cambios no propagaron efectos colaterales no deseados en otras partes del programa.

Pruebas del sistema: Permite validar que el producto final cumpla con los requerimientos del software establecidos por el cliente a nivel funcional, de comportamiento y de rendimiento. Una especificación documentada de los requerimientos del software proporciona la base para el proceso de validación.

Pruebas de aceptación: Estas pruebas las realiza el cliente. Son básicamente pruebas funcionales sobre el sistema completo, y buscan una cobertura de la especificación de requerimientos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, sino una vez pasadas las pruebas de integración por parte del desarrollador.

3. DESCRIPCIÓN DEL CURSO

En el Programa de Bachillerato de Computación e Informática en la Universidad de Costa Rica la ingeniería de software se imparte a través de los cursos Ingeniería de Software I e Ingeniería de Software II, con sus respectivos laboratorios. (Refiérase a la Figura 2).

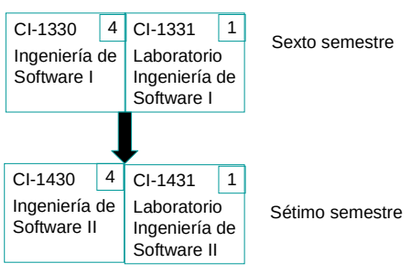

Figura 2. Secuencia de cursos de Ingeniería de Software en el Programa de estudios

Los cursos son semestrales, uno es requisito del otro y cada uno tiene una duración aproximada de 16 semanas lectivas. Específicamente Ingeniería de software I y II son cursos teóricos en donde se les enseña metodologías, técnicas y herramientas de ingeniería de software. Se imparten en 4 horas lectivas semanales y valen 4 créditos cada uno. Por otro lado, en los cursos de laboratorio los estudiantes aplican los conocimientos teóricos, se ofrecen en 2 horas lectivas semanales y valen 1 crédito cada uno.

Durante los dos semestres que dura el curso los estudiantes desarrollan una aplicación comenzando de una pequeña especificación. Al inicio del año se les da una definición básica de los requerimientos, la cual deben completar y mejorar a través de entrevistas y otras técnicas como construcción de programas usando prototipos. Esto hace que cada proyecto, aunque parta de una misma definición, varíe tanto en el diseño de las interfaces de entrada y salida como a nivel de estructuras de datos. El objetivo es fomentar la creatividad en ellos pero con ciertas restricciones, lo cual produce variaciones en el tamaño de las aplicaciones según el equipo que la desarrolló.

Las características de los proyectos son muy similares: el mismo problema a resolver, como plataforma operativa utilizan Microsoft .NET, el lenguaje de programación es ASP.NET y C#, el sistema administrador de base de datos que se utiliza es SQL server, la herramienta de análisis y diseño para hacer el modelado en UML es Rational Rose o StarUML, como herramienta para administrar el proyecto se usa Microsoft Project y para administrar las versiones de software se utilizó Subversion.

Como metodología de desarrollo aplican el Proceso Unificado Racional (RUP, por las siglas en inglés Rational Unified Process), combinado con prácticas de metodologías ágiles tales como Extreme Programming (XP) y Scrum.

El enfoque de administración iterativa de Scrum permite organizar el desarrollo del proyecto a través de pequeñas iteraciones con una duración de 4 a 6 semanas, distribuidas a través de los dos semestres que dura el curso. El uso de RUP durante el desarrollo asegura que en cada iteración se siguen prácticas de calidad en cada una de las fases de desarrollo. La aplicación de XP mejora la construcción de los componentes, garantizando el cumplimiento de los requerimientos y la satisfacción del usuario.

Cada una de cinco fases genéricas del RUP (comunicación, planificación, modelado, construcción y entrega) es guiada por un estándar adaptado a las características del curso, pero siguiendo las prácticas internacionales recomendadas por el IEEE [10] y por PMBOK Guide [18]. En la Tabla 1 se presentan las guías que utilizan los estudiantes para el desarrollo, el estándar internacional que se usó para diseñarlas y la fase en que se aplica.

Tabla 1. Plantillas de desarrollo

Nombre de la plantilla	Fase en la que se aplica
"Guía para elaborar la Descripción Conceptual del Proyecto". Elaborada con base en [18].	Comunicación
"Guía para elaborar planes de administración de proyectos de software". Elaborado con base en [1-10]	Planificación
"Guía para especificar los requerimientos del software". Elaborado con base en [6].	Análisis
"Guía para especificar el diseño del software". Elaborado con base en [4].	Diseño
Las guías de la fase de pruebas son las indicadas en la sección 4.2.2 y fueron elaboradas con base en [5-9].	Pruebas

Cada una de las guías cuenta con su propia plantilla en formato electrónico para facilitar el proceso de documentación.

4. PROCESO DE APLICACIÓN DE LAS TÉCNICAS

4.1 Pruebas de verificación

4.1.1 Participantes

Los roles de los estudiantes en las revisiones técnicas formales son los siguientes:

Equipo de revisión: Equipo de trabajo conformado por los estudiantes del curso, encargado de aplicar las revisiones a otro equipo para validar el cumplimiento del proceso de calidad definido para el curso. Representa al grupo de calidad de una organización. Uno de los miembros debe asumir el rol de líder.

Representante del equipo revisado: Miembro que representa al equipo revisado encargado de evacuar las consultas de los revisores, no debe justificar o defender los defectos encontrados.

4.1.2 Documentos utilizados

Como mínimo el equipo de revisión debe contar con la siguiente información:

- Enunciado de objetivos de la revisión.
- Especificación del elemento del software que se evaluará.
- Elemento del software a revisar.
- Planes, estándares o guías que servirán para revisar el elemento del software.
- Listas de verificación o cotejo para asegurar que el elemento del software cumpla con los requerimientos mínimos.

4.1.3 Procedimiento

Las reuniones para realizar este tipo de revisión se llevan a cabo cuando así lo establezca el calendario del curso o cuando se determine adecuada la revisión del elemento del software por su disponibilidad. El procedimiento a seguir se presenta en la Tabla 2.

Tabla 2. Procedimiento para llevar a cabo una revisión técnica formal

Paso	Descripción
1. Planificación	Líder verifica los criterios de entrada, establece el calendario, identifica y convoca al equipo de revisores y reparte los materiales.
2. Vistazo (opcional)	Si el líder lo considera necesario, convoca a los participantes a una reunión breve en una fecha y hora específica.
3. Preparación	Cada revisor estudia el material en forma individual.
4. Reunión	Durante la revisión el equipo realiza lo siguiente: 1. Revisar el elemento de software contra estándares y guías. 2. Discutir las alternativas o recomendaciones en pro del mejoramiento del elemento de software. 3. Asignar responsabilidades para la solución de problemas encontrados en el elemento de software. Esta responsabilidad puede asignarse al autor o a otros integrantes del proyecto. (Debe hacerse por medio del Líder del Proyecto). 4. Documentar los aspectos técnicos, las recomendaciones y al responsable de resolverlos. 5. Cuando los defectos son numerosos o críticos, el líder debe recomendar un proceso de revisión adicional (re-trabajo).
5. Re-trabajo	Autor revisa y corrige el producto atendiendo la lista de recomendaciones.
6. Seguimiento	Líder comprueba la solución de los problemas realizados por el autor y otros responsables, además de determinar si se necesita realizar otra revisión.

Al finalizar la revisión el profesor evalúa los resultados para asegurar su correctitud y evaluar al equipo revisor. Una vez que los autores revisan y corrigen el producto de software atendiendo la lista de recomendaciones, el profesor realiza una evaluación al producto de software ya corregido.

4.2 Pruebas de validación

4.2.1 Participantes del proceso de pruebas

Los roles de los estudiantes en el proceso de pruebas son los siguientes:

Líder de pruebas: responsable del proceso y especialmente de su planificación, pero también participa del diseño y ejecución de las pruebas pero en menos medida.

Encargados de pruebas: son el resto de los miembros del equipo.

4.2.2 Documentos utilizados

Las plantillas para dar soporte al proceso de pruebas suministradas en el curso fueron diseñadas con base en [5-9] y se listan a continuación:

- **Plan de Pruebas**: Permite registrar la planificación del proyecto de pruebas y contiene información tal como: resumen de los elementos y características que se probarán y las que no se probarán, la estrategia de pruebas, el ambiente de pruebas, los roles y responsabilidades de los grupos encargados de las pruebas y el cronograma del proyecto de pruebas.
- **Diseño de Pruebas**: Tiene dos partes: especificar la estrategia de pruebas y registrar los resultados de las pruebas después de ejecutadas. En la primera parte del formulario se registra información como: el propósito de la prueba, el tipo, las herramientas, los procedimientos (puede referenciar procedimientos automatizados), los datos de entrada (puede referenciar a los casos de prueba) y los criterio de aceptación. En la segunda parte se registra información como: las No Conformidades (NC) con su correspondiente descripción, el estado de la prueba (satisfactoria, fallida, pendiente, cancelada), el responsable, la fecha de ejecución y las incidencias encontradas.
- **Casos de Prueba**: Permite especificar los casos de pruebas. Algunos campos del formulario son: los datos de entrada, el resultado esperado y el flujo que corresponde a la secuencia de pasos para probar la funcionalidad y una descripción de las variables que se están referenciando en el caso de prueba.
- **Reporte de No Conformidades**: Permite registrar las no conformidades clasificadas por tipo, encontradas durante la ejecución de las pruebas: tipo y descripción de la NC, ubicación de la NC, clasificación de importancia, el historial (estado y fecha), respuesta del equipo de desarrollo ante la NC y el responsable.
- **Reporte de Métricas**: Permite registrar las métricas obtenidas en el proceso de pruebas y calcular los indicadores de métricas. Las métricas que se obtienen son: Densidad de fallas, Cobertura de pruebas e Índice de Fallas (o NC) por tipo.

4.2.3 Herramientas de pruebas

Selenium: Es una suite de herramientas de código abierto que se distribuye bajo la licencia Apache License 2.0, y puede descargarse gratuitamente en http://seleniumhq.org. La suite está compuesta por tres herramientas de funcionamiento independiente, a saber: Selenium IDE, Selenium RC y Selenium Grid. Para esta investigación, las herramientas utilizadas fueron las dos primeras las cuales se describen a continuación:

- **Selenium IDE** funciona como un complemento del navegador Mozilla Firefox, y es gracias a esta simbiosis, que le permite grabar toda interacción que el usuario mantiene

de manera natural con un sitio Web, en scripts de casos de prueba, que luego pueden ser reproducidos.

- **Selenium RC** además de proveer la capacidad de ejecutar casos de prueba producidos por Selenium IDE no sólo en el navegador de Mozilla sino en otros (Safari, Google Chrome), permite enriquecer las pruebas, traduciendo los casos de prueba a varios lenguajes de programación de alto nivel.

NUnit: Es una herramienta de código abierto para la ejecución de pruebas de unidad, específicamente en aplicaciones desarrolladas en alguno de los lenguajes de programación del framework de Microsoft .NET, entre ellos: Visual Basic, C++, C# y F#. Es gratuito y puede descargarse desde el sitio www.nunit.org.

4.2.4 Procedimiento

En cada ciclo de pruebas funcionales se realiza el siguiente procedimiento:

1. **Planificación**: Antes de realizar las actividades de validación de requerimientos funcionales el *Líder de pruebas* (como responsable), verifica la disponibilidad de los elementos de prueba y de la documentación correspondiente y procede a planificar las pruebas junto con el resto de los miembros del equipo. En esta fase se utiliza el formulario **Plan de Pruebas**.

2. **Diseño de las pruebas**: El *Encargado de pruebas* de cada funcionalidad o módulo diseña el proceso de pruebas y utiliza los formularios: **Diseño de Pruebas** y **Casos de Prueba**. Una vez completados ambos documentos el *Líder de pruebas* se los envía junto con el Plan de Pruebas al profesor del curso para su revisión. En caso de que los documentos necesiten correcciones, el profesor se lo regresa al *Líder de pruebas* para que realicen las correcciones correspondientes.

3. **Desarrollo del ambiente de pruebas**: EL *Encargado de pruebas* debe crear las pruebas automatizadas para la funcionalidad que le fue asignada. (En el curso ejecutan pruebas unitarias con NUnit y las integran con Selenium IDE para realizar pruebas funcionales.) Posteriormente preparan el ambiente (espacio físico, hardware, software). Si las condiciones no fueran óptimas el *Líder de pruebas* le informa al asistente del curso para que se corrija el problema. Esta información se actualiza en el **Plan de pruebas**.

4. **Ejecución de las pruebas**: El *Encargado de pruebas* procede a ejecutar las pruebas de acuerdo al diseño de pruebas y durante la ejecución registra los resultados en el formulario **Diseño de Pruebas**, incluyendo la imagen que comprueba las NC.

5. **Elaboración de reportes de las pruebas**: Cuando concluye el proceso de pruebas completan los formularios: **Reporte de No Conformidades**, y **Reporte de Métricas**. Este trabajo lo realiza todo el equipo de trabajo.

6. **Evaluación de las pruebas**: Posteriormente el *Líder de pruebas* junto con todos los integrantes del equipo revisan los resultados del ciclo actual de pruebas, y emiten observaciones para que se corrijan los defectos y se repita el proceso en un nuevo ciclo, o se dé por concluido. Si requieren un nuevo ciclo de pruebas deben revisar y actualizar la documentación correspondiente a las pruebas para la nueva ejecución. Si se da por concluido se crea el **Informe Final** para entregar los resultados al profesor.

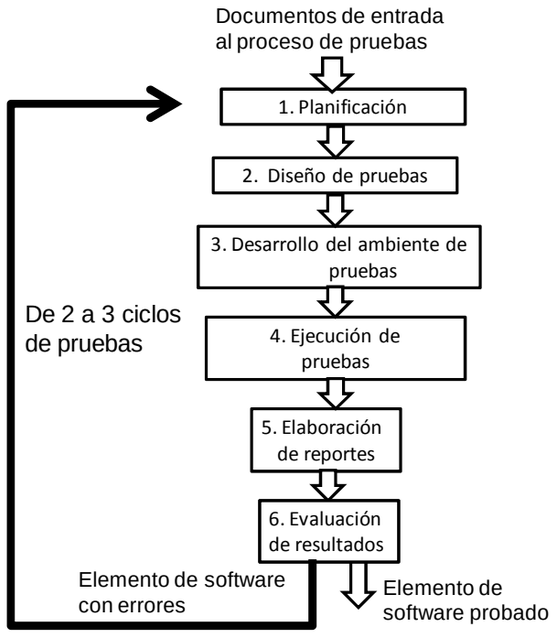

Figura 3. Procedimiento de pruebas.

Al finalizar los tres ciclos de pruebas el profesor con el apoyo del asistente del curso evalúan los resultados para asegurar que efectivamente se cumplen los requerimientos establecidos y analizar los defectos y las métricas reportados.

5. ANALISIS DE RESULTADOS

Durante el año 2010 a los estudiantes se les asignó como proyecto del curso el desarrollo de un sistema web en una arquitectura 4 capas que permitiera administrar los requerimientos de un proyecto de software. Dentro de las funcionalidades solicitadas están las siguientes:

a. Administrar la información básica de los **recursos humanos** que tienen acceso al sistema (administrador, cliente, desarrollador).

b. Administrar la **seguridad** de la aplicación restringiendo el acceso a la información de acuerdo al rol del usuario.

c. Administrar la información básica de un **proyecto,** permitiendo además, crear para cada proyecto el equipo de trabajo (desarrolladores y usuario).

d. Administrar los **requerimientos funcionales** para un determinado proyecto.

e. Administrar los **requerimientos no funcionales** para un determinado proyecto.

f. Para cada requerimiento funcional y no funcional administrar las **dependencias** y los **conflictos** asociados con otros requerimientos del mismo proyecto.

g. Administrar los **cambios** realizados al requerimiento (quién, cuándo, qué y por qué?) y a partir de una línea base controlar las versiones para facilitar el manejo de su historial sin borrar las versiones anteriores.

h. Administrar el **material de apoyo** (casos de uso, casos de pruebas, entre otros) asociados a los requerimientos de un determinado proyecto, de manera que permita el acceso a los archivos electrónicos correspondientes.

i. Administrar el proceso de **verificación y validación de la calidad** de los entregables del proyecto.

j. Algunas **consultas** como por ejemplo un árbol jerárquico que muestre todos los proyectos con sus correspondientes requerimientos ordenados por prioridad y luego por estado.

En este caso de estudio participaron cuatro equipos de trabajo desarrollando la misma aplicación. Al finalizar el desarrollo completo de la aplicación se ejecutaron tres ciclos de pruebas funcionales para validar el cumplimiento de los requerimientos solicitados: el primero y el tercer ciclo lo ejecutaron los mismos autores o desarrolladores de la aplicación a evaluar. El segundo ciclo de pruebas corresponde a una RTF en donde un equipo externo evalúa en forma objetiva una aplicación que no es la suya. El equipo externo es representado por otro equipo de trabajo elegido por el profesor de los mismos estudiantes del curso.

Durante la ejecución de los tres ciclos los estudiantes recolectan medidas para estimar las siguientes métricas: Densidad de fallas, Cobertura de pruebas (por clases y por requisitos) e Índice de fallas por tipo. A continuación se analizan los resultados de las tres métricas recogidas.

5.1 Métrica de densidad de fallas

En la Tabla 3 se muestran los resultados de la métrica Densidad de fallas de los equipos de trabajo del caso de estudio. En la primera columna se identifica el equipo de trabajo, en la segunda columna la cantidad de líneas de código implementadas de la aplicación (LDC), en la tercera, la cuarta y la quinta el número de fallas detectado en los tres ciclos de pruebas correspondientemente y en la sexta, sétima y octava columnas la densidad calculada en los tres ciclos de pruebas correspondientemente.

Tabla 3. Densidad de pruebas.

Equipos de trabajo	KLDC	Fallas ciclo pruebas 1	Fallas ciclo pruebas 2	Fallas ciclo pruebas 3	Densidad ciclo pruebas 1	Densidad ciclo pruebas 2	Densidad ciclo pruebas 3
Equipo 1	12215	10	35	25	0.000818666	0.00286533	0.002046664
Equipo 2	7567	38	65	40	0.008751727	0.01497006	0.009212345
Equipo 3	11230	89	134	97	0.0079252	0.011932324	0.008637578
Equipo 4	10654	5	29	1	0.000469307	0.002721982	9.38615E-05

En la Tabla 3 se puede observar que el ciclo 2 correspondiente a la RTF es donde se obtiene una mayor densidad de fallas, lo que permite afirmar que cuando las pruebas son realizadas por otro equipo de trabajo que no es el que desarrolló la aplicación, se logra detectar un mayor número de fallas. Esto se evidenció en los 4 equipos de desarrollo.

En la Figura 4 se grafica el resultado de las densidades obtenidas permitiendo comparar a los cuatro equipos. Los equipos 1 y 4 lograron una mejor densidad de fallas, especialmente el 4 que en la entrega final la aplicación solo tenía una falla sin corregir.

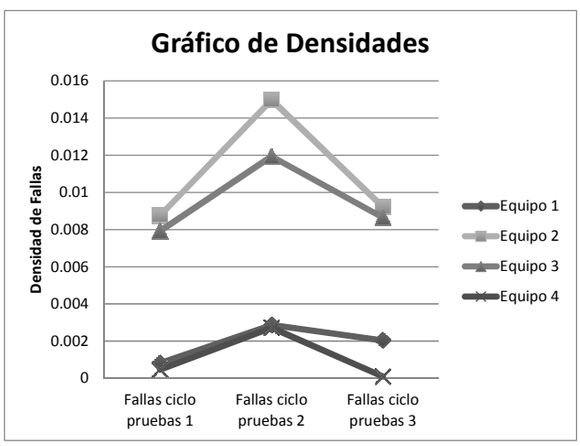

Figura 4. Gráfico de densidades.

5.2 Cobertura de pruebas

En la Tabla 4 se presentan los resultados de la cobertura. Ellos recogieron medidas para dos tipos de métricas de cobertura: la de clases y la de requisitos. En la cobertura de clases solo un equipo dejó de probar una de las clases programadas, logrando un 96% de cobertura. El resto de los equipos tuvo un 100% de cobertura. En cuanto a la cobertura de requisitos de los 25 requisitos establecidos al inicio del curso, solo el equipo 4 cumplió con lo solicitado. El resto de los equipos tiene entre un 89,5% y un 96,5% de cobertura.

Tabla 4. Cobertura de pruebas.

Equipos de trabajo	Clases probadas	Total clases	Cobertura clases	Requisitos probados	Total requisitos	Cobertura requisitos
Equipo 1	25	26	96%	17	25	96.2%
Equipo 2	20	20	100%	17	25	89.5%
Equipo 3	29	29	100%	23	25	92.0%
Equipo 4	26	26	100%	25	25	100.0%

5.3 Distribución de índice de NC por tipo

En la Tabla 5 se presentan los resultados de distribución de las NC por tipo, acumuladas por los 4 equipos de trabajo y recogidas durante los tres ciclos de pruebas. Las NC fueron clasificadas de acuerdo a las siguientes categorías:

1. **Funcionalidad**: Al realizar una acción determinada el resultado que se muestra no está acorde con el esperado.
2. **Validación**: Existen errores relativos a la falta de validación.
3. **Opciones que no funcionan**: Al realizar una acción determinada no se muestra resultado alguno.
4. **Usabilidad**: Se encuentran aquellas inconformidades de efecto visual que provoquen las interfaces de las aplicaciones. Ejemplos: de formato, de visibilidad, de navegabilidad o de correspondencia de etiquetas o títulos mostrados con reales.
5. **Excepciones**: El sistema muestra un mensaje señalando que ha ocurrido un error inesperado o que no ha sido tratado.
6. **No correspondencia de lo implementado con lo documentado:** Incumplimiento de la correspondencia que debe existir entre una aplicación informática y lo que está documentado al respecto.

Tabla 5. Distribución de NC por tipo.

Tipos de Fallas	Funcionales	Validación	Opciones que no funcionan	Usabilidad	Excepciones	Implementado no corresponde con lo especificado
Equipo 1	28	38	4	20	0	0
Equipo 2	40	36	6	30	16	0
Equipo 3	67	22	133	33	16	15
Equipo 4	19	6	0	30	0	4
Total	154	102	143	113	32	19
Distribución porcentu	37%	24%		27%	8%	5%

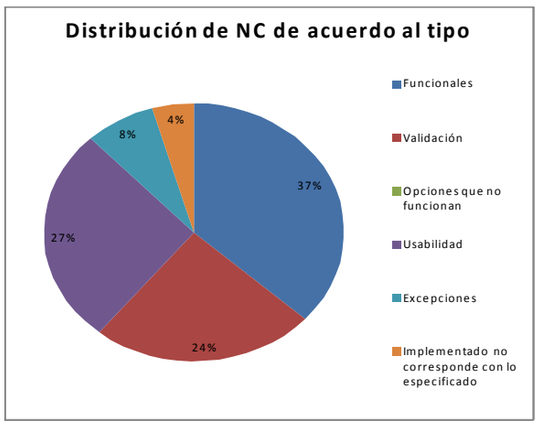

Figura 5. Gráfico de NC por tipo.

En la Figura 5 se muestra que un 37% de las NC ocurre por funcionalidades incorrectas, un 27% por problemas por falta de usabilidad del software y un 24% por una débil validación de las interfaces. No se graficaron las NC del tipo “opciones que no

funcionan" porque de los 143 NC de este tipo, 133 correspondían solamente un equipo de trabajo que entregó la aplicación incompleta.

6. CONCLUSIONES

Con la metodología propuesta se pueden concluir lo siguiente:

- La enseñanza de técnicas de aseguramiento de calidad de software es un área que no debería omitirse en los contenidos de los programas del curso de Ingeniería de Software, debido a que las empresas productoras de software requieren calidad de sus productos para poder competir en mercados internacionales y una forma de apoyarlas es introducir al mercado laboral, profesionales preparados en esta materia.
- La metodología utilizada fue muy positiva porque aprendieron conceptos teóricos de aseguramiento de la calidad del software mediante un enfoque muy práctico.
- Un aspecto que hay que considerar en el proceso de adopción de estándares para el curso, es la necesidad de revisar las exigencias de los estándares del IEEE porque son demasiado específicos y detallistas en algunos puntos, lo que los vuelve poco prácticos y difíciles de implementar. Por esto es necesario primero analizar con cuidado el estándar en forma general y posteriormente hacer una selección de las partes más relevantes para el curso.
- La experiencia de utilizar estándares internacionales en forma sistemática brinda diversos beneficios:
  - Mejora la calidad de los productos que desarrollan los estudiantes.
  - Capacita y disciplina a los estudiantes en buenas prácticas.
  - Proveen una excelente regla contra la cual se les mide la calidad de los entregables implementados en las diferentes fases del ciclo de vida.
- Las revisiones técnicas realizadas entre equipos de trabajo resultan ser muy productivas por las siguientes razones:
  - El equipo revisor actúa con tal rigurosidad y objetividad que detecta errores que generalmente el equipo revisado no encuentra.
  - La retroalimentación que recibe el equipo revisado le permite obtener un producto final más depurado y de mayor calidad.
  - Comparten experiencias, aprenden y/o aclaran aspectos técnicos que no habían considerado en sus proyectos o que no tenían claros.
- El impartir este curso no es una tarea fácil debido a que abarca una amplia variedad de temas de ingeniería de software. Sin embargo, en general los estudiantes manifiestan mucho interés y satisfacción al aprender en forma práctica todo el conocimiento adquirido.
- Las métricas recogidas durante el proceso de pruebas permiten conocer el estado del mismo y mejorar las deficiencias encontradas.

7. RECONOCIMIENTO

Se agradece a los estudiantes del curso de Ingeniería de Software del año 2011 su colaboración en esta labor de investigación porque el trabajo de ellos permitió probar estas metodologías.

8. REFERENCIAS

[1] IEEE Computer Society. 1987. ANSI/IEEE Std. 1058.1-1987, IEEE Standard for Software Project Management Plans.

[2] IEEE Computer Society. 1988. IEEE Std 982.1-1988 Standard Dictionary of Measures to Produce Reliable Software.

[3] IEEE Computer Society. 1990. IEEE Std 610.12.1990 Standard Glossary of Software Engineering Terminology.

[4] IEEE Computer Society. 1993. Std 1016.1-1993 IEEE Guide to Software Design Descriptions –Description.

[5] IEEE Computer Society. 1998. IEEE Std 982.1-1988 Standard Dictionary of Measures to Produce Reliable Software.

[6] IEEE Computer Society. 1998. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications –Description.

[7] IEEE Computer Society. 1998. IEEE Std 1012.1998. Standard for Software Verification and Validation.

[8] IEEE Computer Society. 1998. IEEE Std 1061-1998 Standard for a Software Quality Metrics Methodology, (Revision of IEEE Std 1061-1992).

[9] IEEE Computer Society. 1998. IEEE Std 829.1998 Standard for Software Test Documentation.

[10] IEEE. 2003. IEEE Standards Collection: Software Engineering. IEEE Inc. 2003. ISBN: 978-0738137575.

[11] ISO. 1991. ISO 9000-3 Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software. ASQC Press.

[12] Kaner, C.; Falk, J. & Nguyen, H.1999. Testing Computer Software. Wiley, second edition, USA.

[13] Kaner, C., Bach, J. & Bred, P.. 2002. Lessons Learned in Software Testing. Wiley, USA.

[14] Lewis, W.. 2009. Software Testing and Continous Quality Improvement. 3rd ed, CRC Press.

[15] Myers, G.. 1979. The Art of Software Testing. Wiley-Interscience.

[16] Paulk, M.; *et al*. 1995. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley.

[17] Pressman, R. 2010. Ingeniería de Software: Un enfoque práctico. Séptima edición, McGraw-Hill.

[18] Project Management Institute. 2005. Guide to the Project Management Body of Knowledge (PMBOK Guide). Tercera Edición. ISBN: 1-880410-23-0.

[19] Quality Assurance Institute. 2006. CSTE Common Body of Knowledge. V6.2.

[20] SEI. SOFTWARE ENGINEERING INSTITUTE. 2006. CMMI for Development (CMMI-DEV), Versión 1.2 Technical report CMU/ SEI-2006-TR-008. Pittsburg, PA: Software Engineering Institute, Carnegie Melon University.

[21] Yourdon E.. 1993. Decline and Fall of the American Programmer. Yourdon Press.