UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

GUIA DE METRICAS BASICAS DE SOFTWARE

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRA EN CIENCIAS

PRESENTA:

CRISTINA PATRICIA DAVILA VEGA

DIRECTORA DE LA TESIS: DRA. HANNA OKTABA

MEXICO, D.F. **2002**

Quiero dedicar especialmente esta tesis a mi querida hija

Nancy Abigail

porque su llegada en el transcurso de la realización de este trabajo ha sido la alegría más grande de mi vida y su compañía es una luz de amor en el hogar de todos ustedes.

AGRADEZCO:

*Especialmente a la **Dra. Hanna Oktaba** por haber aceptado ser mi asesor de tesis, por guiarme y apoyarme de la mejor manera para hacer mi trabajo y también por su amable paciencia y atención que siempre me brindó. ¡Muchas Gracias!*

*A mi **esposo** por todo su apoyo, amor y comprensión que nunca me faltaron.*

*A mis **padres** y **hermanos** por estar siempre al pendiente de mí brindándome su ayuda incondicional, especialmente a mi hermana **Mary** que siempre me acompañó a todos lados cuidando de mi hija.*

*A cada uno de mis sinodales, **Maestra Lupita, Dr. Fernando Gamboa, Maestro Jorge Ortega y Dr. Manuel Romero**; por la atención tan amable que siempre han tenido conmigo.*

*A **Lulú González** por brindarme todas las facilidades y orientación en los trámites que tuve que realizar.*

INDICE

Pág.

INTRODUCCIÓN

Los desarrolladores de software, aún cuando sean muy competentes, cometen muchos errores al realizar un sistema. Estos errores son defectos que podrían ser difíciles de encontrar. Si en forma individual un ingeniero tiene este tipo de problemas, lo más probable es que la empresa a la que pertenece también los tenga, ya que la calidad de un producto de software inicia desde el trabajo individual de cada uno de los miembros del equipo de trabajo.

Entre más grande sea un sistema, la dificultad de encontrar y corregir los defectos se incrementa. Por tal motivo, es importante que se elaboren trabajos de alta calidad desde el inicio del proyecto. De esta manera, es posible incrementar la productividad en forma individual y, al mismo tiempo, de toda la empresa.

Sabemos que existe un proceso disciplinado personal de Ingeniería de Software: el PSP [8] (Personal Software Process), descrito por Humphrey, el cual consiste en mostrar y explicar métodos del manejo efectivo del tiempo y defectos, así como la planeación, el seguimiento y el análisis de los mismos.

El presente trabajo propone dos patrones cuyo objetivo es ayudar a obtener métricas básicas de software con respecto al tiempo y los defectos. Dichas métricas están contenidas en el PSP, y pueden servir para mejorar la productividad del equipo de trabajo, y por ende, la calidad de los proyectos que se realicen.

OBJETIVOS

- Proporcionar una guía clara y concisa para obtener métricas básicas de software como son tiempos totales de producción, pruebas y correcciones; así como indicadores de defectos encontrados y corregidos. Por ejemplo: números totales de defectos encontrados, corregidos, por tamaño y por tiempo, por mencionar algunos.

- Representar la guía mediante patrones que expresen el proceso para obtener los indicadores mencionados anteriormente.

- Lograr que los mismos patrones de la guía apoyen el diseño preliminar para una herramienta de recolección de métricas básicas y cálculo de indicadores para los proyectos de software.

METODOLOGÍA

La Guía de Métricas Básicas de Software fue elaborada en base a cuatro patrones, cuya autoría es la Doctora Hanna Oktaba y la Maestra Guadalupe Ibargüengoitia. Dichos patrones son:

1. **Patrón de registro del tiempo de una actividad,** consiste en contar y registrar el tiempo que se invierte en las actividades del proceso de software.

2. **Patrón de registro del tamaño del producto,** calcula el tamaño de los productos de software.

3. **Patrón de registro de búsqueda de defectos,** registra los defectos encontrados en los productos de software.

4. **Patrón de registro de corrección de defectos,** registra la información sobre los defectos corregidos en los productos de software.

Los indicadores sobre defectos de PSP [8] (Personal Software Process) que se calculan en los patrones propuestos en esta tesis, se eligieron al realizar un análisis de los libros **Introduction to the Personal Software Process e Introduction to the Team Software Process,** ambos del autor Watts S. Humphrey. En estos textos se mencionan los siguientes indicadores:

INDICADORES DE PSP	INDICADORES SELECCIONADOS PARA SU ANÁLISIS EN ESTA GUÍA
Número total de defectos	✓
Número de defectos por tipo (encontrados y/o corregidos)	✓
Número de defectos por líneas de código o por tamaño	✓
Número de defectos por rendimiento	
Proporción de defectos introducidos	
Proporción de defectos eliminados o corregidos	
Proporción de tiempo invertido en la búsqueda de defectos	
Número de defectos encontrados pero no corregidos	✓
Número de defectos encontrados por hora	✓
Número de defectos corregidos por hora	✓
Número de defectos encontrados por hora de pruebas	✓
Número de defectos corregidos por hora de correcciones	✓

En la parte derecha de la tabla, se señalan con un símbolo de palomita los indicadores que se calculan en uno de los patrones propuestos en esta tesis. Los indicadores que no fueron seleccionados se pueden calcular con la información obtenida de los patrones incluidos en esta guía. Los defectos por rendimiento se refieren principalmente al porcentaje de

defectos encontrados por un método de eliminación, y se puede hacer el cálculo ya sea por proceso o por fase, aplicando las siguientes fórmulas:

- 100 * (defectos eliminados antes de compilar) / (defectos introducidos antes de compilar) ó

- 100 * (defectos eliminados durante la fase) / (defectos en el producto en la entrada de la fase) respectivamente.

La proporción de defectos introducidos es de utilidad para estimar cuántos defectos se cometen en cada fase. Solamente se multiplica la proporción esperada de introducción de defectos por el tiempo estimado a ser invertido en dicha fase. De la misma forma podemos calcular la proporción de defectos corregidos. Y la proporción del tiempo invertido en la búsqueda de defectos se obtiene del registro de tiempo invertido en la actividad de búsqueda de defectos.

La mejor manera de presentar la obtención de dichos indicadores es en la forma de patrón, en lugar de una descripción en lenguaje natural, ya que es una forma generalmente conocida por los ingenieros de software.

El primer patrón se llama **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones** y calcula los tiempos reales dedicados a la producción, pruebas y correcciones de un producto de software desde su creación inicial hasta su primer ingreso bajo el control de configuración. Utiliza los patrones **Registro de Tiempo de una Actividad, Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos,** toma los tiempos recolectados por estos patrones para realizar una sumatoria y así obtener los tiempos totales.

El segundo patrón se llama **Patrón de Indicadores sobre Defectos Encontrados y Corregidos** y calcula el número total de: defectos encontrados, corregidos, encontrados y no corregidos, encontrados y corregidos, por tamaño, encontrados por hora de pruebas, y corregidos por hora de correcciones. Utiliza los patrones **Registro de Tamaño del Producto, Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos,** tomando el registro de defectos y el tamaño del producto para hacer los cálculos necesarios.

CONTENIDO

La presente tesis está dividida en dos partes: un marco teórico y el desarrollo de los patrones.

La primera parte incluye los capítulos 1, 2 y 3. En los siguientes párrafos se explica brevemente el contenido de cada uno de ellos.

El Capítulo 1 trata sobre los fundamentos de medición, como por ejemplo, la definición de métrica, los objetivos de la medición del software, la relación con respecto al control y mejora del software, el alcance de las métricas y su clasificación.

El Capítulo 2 explica en qué consiste el Proceso Personal de Software, las métricas que contiene, cómo se lleva a cabo el manejo del tiempo y el control de defectos.

El Capítulo 3 explica de una manera concisa qué es UML, qué significan cada uno de los diagramas de los que consta y una descripción de cada uno de sus elementos; así como una breve explicación de los conceptos básicos de patrones.

La segunda parte consta de los capítulos 4, 5, 6 y 7. En los siguientes párrafos se explica brevemente el contenido de cada uno de ellos.

El Capítulo 4 ofrece una introducción a la segunda parte mencionando el objetivo de esta guía, su contenido y muestra también, un diagrama de clases que hace más comprensibles los patrones desarrollados.

El Capítulo 5 presenta la traducción al español de los cuatro patrones básicos de métricas [10] desarrollados por la Doctora Hanna Oktaba y la Maestra Guadalupe Ibargüengoitia.

Los patrones de los capítulos 6 y 7 se basan en los del capítulo 5. Se decidió incluir los de este capítulo en su forma íntegra, traducidos al español, como parte de este trabajo.

Los Capítulos 6 y 7 presentan el **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones**, y el **Patrón de Indicadores sobre Defectos Encontrados y Corregidos**, respectivamente.

De esta manera, la segunda parte incluye los capítulos 4, 5, 6, y 7, éstos pueden considerarse como guía íntegra de métricas básicas de software basada en PSP [8].

Al final se presentan las conclusiones.

PARTE 1 MARCO TEÓRICO

CAPÍTULO 1 FUNDAMENTOS SOBRE LA MEDICIÓN

Las métricas de software se han convertido en un punto esencial en la Ingeniería de Software, ya que los desarrolladores miden ciertas características del software con el propósito de obtener información, en el sentido de que si los requerimientos son consistentes y completos, o si el diseño es de alta calidad, o simplemente si el código está listo para ser probado. Un administrador de proyectos debe medir los atributos de los procesos y de los productos para que de esta manera pueda estimar con más certeza cuándo el software estará listo para su liberación, si los costos excedieron lo que se tenía presupuestado, si los pasos a seguir de un proceso son efectivos y se pueden mejorar.

Realizamos mediciones para reunir datos que necesitamos para obtener un entendimiento cuantitativo, para evaluar un producto, proceso u organización, para controlar un producto o proceso y para hacer un plan.

En este capítulo se explican algunos fundamentos sobre medición, empezando con una definición de lo que es una métrica, cuáles son sus objetivos, su importancia en el proceso de mejora de software y cómo se clasifican.

1.1 ¿QUÉ ES UNA MÉTRICA?

Los términos "medida", "medición" y "métrica" son diferentes; dentro del contexto de la Ingeniería de Software, una medida proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto.

La medición es el acto de determinar una medida. Podemos definir formalmente medición como "el proceso por el cual se asignan números o símbolos a los atributos de un proceso o producto, de tal manera que los describan de acuerdo a reglas claramente definidas." [5]

Un atributo es una característica o propiedad de un proceso o producto. A menudo definimos atributos usando números o símbolos, los cuales son muy útiles e importantes, ya que permiten realizar juicios sobre los procesos o productos con solo conocer y analizar sus atributos.

La medición proporciona una manera sistemática de aprender de las experiencias pasadas y aplicarlo a las actividades actuales.

El "IEEE Standard Glossary of Software Engineering Terms" define métrica como "una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado" [11]. Una métrica, por sí sola, no representa nada. Es un número que debe ser comparado con alguna norma o estándar para que tenga un significado útil.

Muchas actividades involucran algún grado de medición de software por ejemplo:

- estimación de costo y esfuerzo.
- métricas y modelos de productividad.
- recolección de datos.
- modelos y métricas de calidad.
- modelos de confiabilidad.
- evaluación y modelos de ejecución.
- métricas estructurales y de complejidad.
- evaluación de la madurez de la capacidad.
- administración por métricas.
- evaluación de métodos y herramientas.

1.2 OBJETIVOS DE LA MEDICIÓN DEL SOFTWARE

La medición es necesaria para valorar el estado de nuestros proyectos, productos, procesos y recursos, para que de esta manera se tenga un mayor control.

Cada medición debe ser motivada por una meta o necesidad particular que esté claramente definida y fácilmente entendible. Los objetivos de las métricas deben ser específicos y de acuerdo a lo que los administradores, desarrolladores y usuarios necesitan conocer. Estos objetivos pueden diferir de acuerdo al tipo de personal involucrado, van a indicar cómo será utilizada la información de medición una vez que ha sido recolectada [5]. El tipo de información que se necesita para entender y controlar un proyecto de software depende de la perspectiva que se vea, ya sea un administrador o un ingeniero.

Los administradores necesitan saber:

- Cuánto cuesta cada proceso.

- Qué tan productivo es el equipo de trabajo.
- Qué tan bien se está desarrollando el código.
- Si el usuario estará satisfecho con el producto.
- Cómo se puede mejorar.

Los ingenieros necesitan saber:

- Si los requerimientos fueron probados.
- Si se han encontrado todas las fallas.
- Si se han obtenido las metas del producto o proceso.

1.3 LA MEDICIÓN PARA EL ENTENDIMIENTO, CONTROL Y MEJORA

La medición es importante para tres actividades básicas. Primero, hay métricas que ayudan a **entender** qué esta sucediendo durante el desarrollo y mantenimiento; se valora la situación actual, estableciendo líneas base que ayuden a definir objetivos para sucesos futuros; en este sentido, las métricas hacen más visible los aspectos de un proceso y producto, con lo que se puede entender mejor la relación entre las actividades y las entidades que los afectan.

Segundo, las métricas permiten **controlar** lo que está sucediendo en los proyectos, utilizando líneas base, objetivos y entendimiento de las relaciones, se predice lo que es probable que suceda y así poder hacer cambios a los procesos y productos que ayudan a obtener las metas.

Tercero, las métricas alientan para **mejorar** los procesos y productos.

Es importante manejar las expectativas de quienes tomarán las decisiones basadas en estas mediciones. Los usuarios de los datos deben siempre estar concientes de la exactitud limitada de la predicción y del margen de error en las mediciones.

1.4 CLASIFICACIÓN DE MÉTRICAS DE SOFTWARE

El primer objetivo de cualquier actividad de medición de software es identificar las entidades y atributos que se desean medir. En el contexto de software hay tres clases de estas entidades:

Procesos, que son colecciones de actividades de software que están relacionadas.

Productos, que son cualquier artefacto, entregables o documentos que resultan de una actividad de un proceso.

Recursos, que son entidades requeridas por una actividad de un proceso.

Casi siempre un proceso es asociado con alguna escala de tiempo, es decir, las actividades de un proceso son ordenadas o relacionadas de alguna manera que dependen del tiempo, así una actividad debe ser completada antes de que otra pueda iniciar.

Los recursos y los productos son asociados con el proceso. Cada actividad del proceso tiene recursos y productos que utiliza, así como productos que son generados. Por este motivo, el producto de una actividad puede ser la entrada de otra actividad.

1.4.1 Métricas del Producto

Las métricas del producto generalmente se refieren al volumen del producto generado. Incluyen líneas de código, páginas de un documento, número de pantallas, número de archivos, etc. Estas métricas pueden ser de varios elementos del producto, por ejemplo, la cantidad de código producido en la fase de implementación o las líneas de código modificadas durante la prueba de unidad.

1.4.2 Métricas del Proceso

Las métricas del proceso cuantifican la conducta del proceso; una categoría principal de ellas es el conteo de eventos, por ejemplo, se cuentan los elementos que ocurren, tal como los números de defectos encontrados en la prueba o los cambios en los requerimientos.

Otra categoría son las métricas de tiempo; muy frecuentemente se le llama tiempo de ciclo al tiempo requerido para completar un proyecto. Tener un conocimiento preciso del tiempo que se requiere para llevar a cabo las diversas tareas de un proyecto puede ayudar a optimizar los procesos de acuerdo a los recursos, al tiempo de ciclo o a alguna combinación de los dos. Estas métricas también ayudan a realizar mejores planes de desarrollo.

1.4.3 Métricas de los Recursos

Las métricas de recursos se refieren a las horas laborales. Esto implica a las horas de trabajo, categorías de trabajo y actividades de las tareas. Mientras las métricas comunes de recursos están programadas en meses o semanas, las métricas de tiempo personales son muy útiles si están en minutos.

Cuando se reúnen datos detallados de tiempo, se pueden identificar muchas actividades de interrupción que en la mayoría de las ocasiones se le dedica mucho tiempo y de esta

manera se reduce la productividad. Por lo que se considera que un foco principal de mejora de la productividad y del tiempo de ciclo es identificar y reducir estas distracciones. Pero no es suficiente con decirle a la gente que reduzca esas interrupciones, se requiere un estudio para saber en qué están invirtiendo su tiempo y qué se podría hacer para reducir las distracciones.

El presente trabajo se enfoca principalmente a las métricas del producto, también se trabaja con las métricas de tiempo de las actividades que se llevan a cabo para generar un producto y con algunas métricas del proceso como son el conteo del número de defectos encontrados y corregidos. Así de esta manera, se puede entender y controlar lo que está sucediendo durante el desarrollo de un proyecto y hacer mejoras en los procesos y productos.

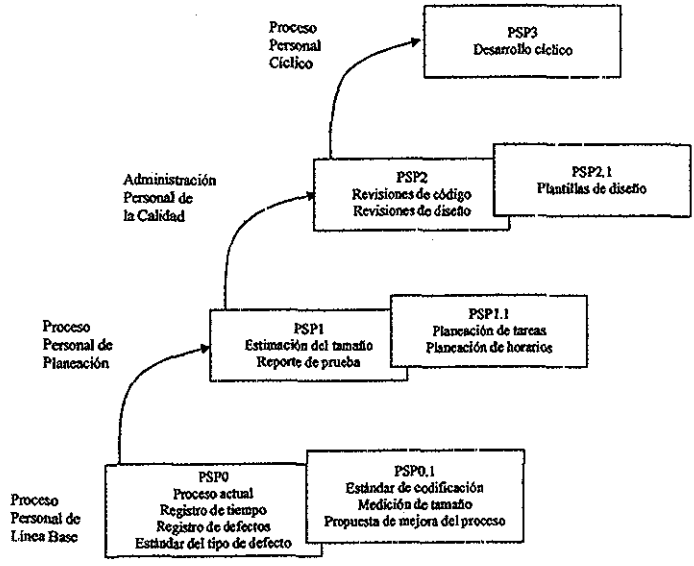
Para presentar la **Guía de Métricas Básicas de Software** se ha tomado como base las mejores prácticas relacionadas con este tema incluidas en el Proceso Personal de Software, (Personal Software Process) PSP [7].

CAPÍTULO 2 EL PROCESO PERSONAL DE SOFTWARE (PSP)

El Proceso Personal de Software (PSP) [7] tiene como una parte fundamental la reunión y uso de los datos. Una parte principal de esta **Guía de Métricas Básicas de Software** es reunir datos y luego utilizarlos para obtener información que sea útil para la toma de decisiones. Este capítulo describe brevemente qué es el PSP y qué elementos relacionados con las mediciones de tiempo y defectos maneja.

El PSP es un proceso de mejora diseñado para controlar, manejar y mejorar la manera de desarrollar software. Es una serie estructurada de formas, guías y procedimientos. Si el PSP se utiliza de forma apropiada proporciona datos históricos que se necesitan para hacer mejor el trabajo y que éste sea más predecible y eficiente.

PSP está definido por capas [5] y a continuación se resumen las actividades y mediciones que se introducen gradualmente en la siguiente figura:



La Evolución de PSP

2.1 PSP0: EL PROCESO DE LÍNEA BASE

El primer paso en el PSP es establecer una línea base que incluye algunas mediciones básicas de tiempo y defectos y un formato de reporte. Esta línea base proporciona una referencia consistente para medir el progreso y un fundamento definido sobre el cual mejorar. PSP0 debe ser el proceso que se usa para hacer software pero con la diferencia de que éste proporciona mediciones de tiempo y defectos [8].

PSP0 es extendido en PSP0.1 al cual se le agregan estándares de codificación, métricas de tamaño y el propósito de mejora del proceso (PIP) que es un formato que proporciona una manera estructurada de registrar los problemas, experiencias y sugerencias de mejora del proceso [8].

2.2 PSP1: EL PROCESO PERSONAL DE PLANEACIÓN

PSP1 contiene pasos de planeación basados en PSP0, se agrega un reporte de prueba y una estimación de tamaño y recursos. En PSP1.1. se introducen la planeación de tareas y horarios. Las razones por las cuales son útiles estos planes son los siguientes [8]:

- Para ayudar a entender la relación entre el tamaño de los programas y el tiempo que se invierte en desarrollarlos.
- Para llevar a cabo el trabajo de forma ordenada.
- Para que proporcione un medio para determinar el estado del trabajo

Estos puntos son críticos tanto para un proyecto muy grande como para uno pequeño. Una vez que se conoce el desempeño, podemos planear el trabajo de manera más exacta, hacer metas más realistas y satisfacerlas de forma más consistente.

2.3 PSP2: EL PROCESO PERSONAL DEL MANEJO DE LA CALIDAD

Uno de los objetivos del PSP es ayudar a comprender de forma realista y objetiva que los defectos de los programas son resultado de los errores de los mismos desarrolladores[8].

Se deben mejorar las habilidades y capacidades para hacer trabajos de calidad. Para manejar los defectos es necesario conocer cuántos se cometen. En PSP2 se agregan técnicas de revisión para encontrar defectos a tiempo, es decir, antes de que se conviertan en defectos caros de corregir; se reúnen y analizan y de esta manera se puede establecer listas de verificación de revisión y hacer evaluaciones de calidad del proceso.

PSP2.1 se enfoca al proceso de diseño, establece criterios para determinar qué tan completo está y examina diversas verificaciones de diseño y técnicas de consistencia [8].

2.4 PSP3: UN PROCESO CÍCLICO PERSONAL

Hay programas que son demasiado largos y que sería muy difícil hacer una revisión utilizando PSP2 por la cantidad de tiempo que se invertiría. Para estos casos, se puede utilizar PSP3 cuya estrategia consiste en subdividir un programa en piezas, donde en cada una de ellas se aplica PSP2 de forma iterativa.

2.5 MANEJO DEL TIEMPO SEGÚN PSP

Para manejar el tiempo de un ingeniero de software de manera efectiva, es necesario que se haga un plan y se apegue a él. Este plan tiene que reflejar las actividades que se realizan y el tiempo que se invierte en ellas. Teniendo esta documentación, se pueden mejorar los planes futuros haciendo comparaciones con los anteriores en los cuales se detectarán los errores de estimación que se cometieron.

Para practicar el manejo del tiempo se tienen que considerar los siguientes pasos:

- Clasificar las actividades.
- Registrar el tiempo invertido en cada actividad.
- Registrar el tiempo de una manera estandarizada.
- Mantener los datos de tiempo en un lugar conveniente.

2.6 SEGUIMIENTO DEL TIEMPO

La manera de mejorar la calidad de trabajo es que se conozcan las tareas que se hacen, cómo se hacen y el resultado que se obtiene. El primer paso en este proceso es definir las tareas y medir cuánto tiempo se invierte en cada una de ellas.

Al registrar el tiempo hay que recordar que el objetivo es obtener los datos verídicos sobre la duración de las tareas. El formato y procedimiento que se use para reunir estos datos no es tan importante como que los datos sean exactos y completos.

Además de conocer en qué se invierte el tiempo, también se necesita llevar un seguimiento de los resultados producidos. De esta manera se podría calcular la productividad de la tarea.

2.7 MANEJO DE DEFECTOS SEGÚN PSP

Los defectos pueden causar problemas muy serios a los usuarios de los productos de software y puede ser muy costoso encontrarlos y corregirlos. Los defectos son errores que los desarrolladores cometen, por lo que necesitan estar concientes de los que introducen y aprender a manejarlos. El primer paso para manejar los defectos es reunir datos sobre los defectos introducidos en los programas, de esta manera se puede mejorar la manera de encontrarlos y corregirlos.

Los defectos pueden encontrarse en los programas, diseños, requerimientos, especificaciones u otro tipo de documentación. Un defecto es algo objetivo que se puede identificar, describir y contar.

Un problema muy importante que ocasionan los defectos es que lleva mucho tiempo y dinero encontrarlos y corregirlos. Para cometer menos defectos se debe aprender de los que ya se han introducido, identificando los errores que los causaron para evitar repetir el mismo error en el futuro.

Para reunir datos sobre defectos en un programa, se puede hacer lo siguiente:

- Mantener un registro de cada uno de los defectos que se encuentren en un programa.
- Registrar suficiente información sobre cada defecto para que no haya confusiones.
- Analizar estos datos para ver qué tipo de defectos causaron más problemas.
- Proponer formas de encontrar y corregir estos defectos.

Aunque no hay manera de evitar la introducción de defectos, es posible encontrar y remover casi todos ellos desde el inicio del desarrollo. Si se remueven los defectos lo más pronto posible, se ahorra tiempo y se generan mejores productos. Hay muchas maneras de encontrar defectos en un programa, estos métodos implican los siguientes pasos:

1. Identificar los síntomas de cuando ocurre un defecto.
2. Deducir de estos síntomas la ubicación del defecto.
3. Resolver cuál es el error en el programa.
4. Decidir cómo corregir el defecto.
5. Hacer la corrección.
6. Verificar que la corrección ha resuelto el problema.

La métrica básica de defecto utilizada en el PSP es la de número de defectos por líneas de código (LOC).

Otras métricas que podemos obtener con los datos de defectos son:

- Defectos encontrados.

- Defectos corregidos.
- Defectos encontrados por hora.
- Defectos corregidos por hora.

La presente tesis se basa en las métricas que define y maneja PSP, específicamente las métricas de tiempo de las actividades y las métricas de defectos encontrados y corregidos.

Básicamente, las métricas que se usaron para elaborar esta guía son:

- El tiempo total invertido en la actividad de producción.
- El tiempo total invertido en la actividad de pruebas.
- El tiempo total invertido en la actividad de correcciones.
- El tiempo total invertido en el producto desde su producción hasta su ingreso al control de versiones.

También se usaron los siguientes indicadores de defectos:

- Número de defectos encontrados y no corregidos.
- Número de defectos por tamaño del producto.
- Número de defectos encontrados por hora en base al tiempo total del producto.
- Número de defectos corregidos por hora en base al tiempo total del producto.
- Número de defectos encontrados por hora en base al tiempo total de la actividad de pruebas.
- Número de defectos corregidos por hora en base al tiempo total de la actividad de correcciones.

Las métricas de tiempo total invertido en las actividades de producción, pruebas y correcciones pueden ser muy útiles para que en proyectos futuros se realicen propuestas en cuanto a mejoras en las técnicas de producción, pruebas y correcciones y después, se evalúe de manera cuantitativa el efecto que tuvieron, ya sea positivo o negativo.

Los indicadores de defectos encontrados y corregidos nos pueden servir para realizar estimaciones más certeras en cuanto a tiempo, a recursos y principalmente en cuanto a número de defectos que se introducirán y removerán en futuros proyectos.

Tanto las métricas de tiempo como los indicadores de defectos nos ayudan a mejorar, en gran medida, la calidad del producto y la de los procesos que se utilizan para su producción y nos proporcionan una base para la planeación de proyectos futuros.

CAPITULO 3 UNIFIED MODELING LANGUAGE (UML) Y PATRONES

En el presente capítulo se describe en qué consiste el lenguaje de modelado unificado UML [1], se definen los conceptos básicos que forman parte de él y también se explica qué es un patrón y las secciones de las que consta. El motivo por el cual se trata UML en este capítulo es entender la representación gráfica que se hace de los patrones que se proponen en esta **Guía de Métricas Básicas de Software**.

La teoría sobre patrones es una parte importante de este trabajo ya que se descubren dos patrones. Uno calcula el tiempo total de las diferentes actividades que se llevan a cabo para generar un producto y el otro, obtiene información sobre los defectos encontrados y corregidos durante dichas actividades.

3.1 UML

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar los productos de un sistema de software [1].

En un sistema de software hay elementos que se pueden modelar mejor textualmente y hay otros, que pueden ser modelados mejor gráficamente y UML es uno de los lenguajes gráficos que puede ser de utilidad en estos casos. Detrás de cada símbolo que forma parte de la notación de UML existe una semántica bien definida, por lo que un desarrollador puede realizar un modelo en UML y otro puede interpretarlo sin mayores problemas, aún cuando este último maneje otra herramienta de modelado.

UML es un lenguaje gráfico para **especificar**, esto significa construir modelos que sean precisos, sin ambigüedad y completos. En particular, UML conlleva la especificación de todas las decisiones importantes de análisis, diseño e implementación que se deben hacer en el desarrollo y ejecución de un sistema de software.

UML no es un lenguaje visual de programación, pero sus modelos pueden ser traducidos a muchos lenguajes de programación, tales como Java, C++ o Visual Basic.

Para entender UML es necesario formar un modelo conceptual del lenguaje, para lo cual son importantes tres elementos:

- Los bloques básicos de construcción de UML.

- Las reglas que dictan cómo unir estos bloques.
- Los mecanismos comunes que se aplican en UML.

En UML existen tres tipos de bloques de construcción:

1. Elementos.
2. Relaciones.
3. Diagramas.

Los elementos son abstracciones y son los componentes principales en un modelo; las relaciones unen a estos elementos y los diagramas agrupan colecciones de elementos.

Hay cuatro tipos de elementos en UML:

1. Estructurales.
2. De comportamiento.
3. De agrupación.
4. Notacionales.

Los **elementos estructurales** son los nombres de los modelos de UML y son las partes estáticas de un modelo, ya que representan elementos conceptuales o físicos. Hay siete tipos de elementos estructurales [1]:

1. **Clase:** descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

2. **Interfaz:** colección de operaciones que especifican un servicio de una clase o componente.

3. **Colaboración:** define una interacción, es una sociedad de roles y otros elementos que trabajan juntos para proporcionar algún comportamiento cooperativo que sea más grande que la suma de todos los elementos.

4. **Caso de uso:** descripción de un conjunto de secuencia de acciones que un sistema ejecuta y produce un resultado observable para un actor en particular.

5. **Clase activa:** clase cuyos objetos abarcan uno o más procesos o hilos de control y que pueden iniciar una actividad de control.

6. **Componente:** parte física y reemplazable de un sistema que conforma y proporciona la realización de un conjunto de interfases.

7. **Nodo:** elemento físico que existe en tiempo de corrida y representa un recurso computacional que generalmente tiene al menos memoria y capacidad de procesamiento.

Los elementos de comportamiento son las partes dinámicas de los modelos de UML, son los verbos de un modelo que representan comportamiento sobre tiempo y espacio. Hay dos tipos principales de elementos de comportamiento [1]:

1. **Interacción:** comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto particular para lograr un propósito específico.

2. **Máquina de estado:** comportamiento que especifica las secuencias de estados de un objeto o una interacción y que siguen a través de su tiempo de vida en respuesta a eventos.

Los elementos de agrupación son las partes organizacionales de los modelos de UML. Solo hay un tipo principal de elemento de agrupación llamado **paquete**, que es un mecanismo de propósito general para organizar elementos dentro de grupos.

Los elementos notacionales son las partes explicativas de los modelos de UML. Hay un tipo principal de elemento notacional llamado **nota**, la cual es simplemente un símbolo para interpretar restricciones y comentarios unido a un elemento o una colección de elementos.

Hay cuatro tipos de relaciones en UML [1]:

1. **Dependencia:** relación semántica entre dos elementos en la cual un cambio de un elemento puede afectar la semántica de los demás elementos.

2. **Asociación:** relación estructural que describe un conjunto de ligas, entendiendo como liga una conexión entre objetos.

3. **Generalización:** relación de especialización/generalización en la que los objetos del elemento generalizado son sustituibles por objetos del elemento especializado.

4. **Realización:** relación semántica entre clasificadores, donde un clasificador especifica un contrato que otro clasificador garantiza llevar a cabo.

Un **diagrama** es la presentación gráfica de un conjunto de elementos, sirven para visualizar un sistema desde diferentes perspectivas. UML cuenta con nueve diagramas [1]:

1. **Diagrama de clases:** muestra un conjunto de clases, interfases y colaboraciones y sus relaciones.

2. **Diagrama de objetos:** muestra un conjunto de objetos y la comunicación entre ellos.

3. **Diagrama de casos de uso:** muestra un conjunto de casos de uso y actores y sus relaciones.

4. **Diagrama de secuencia:** diagrama de interacción que enfatiza el orden de tiempo de mensajes.

5. **Diagrama de colaboración:** diagrama de interacción que enfatiza la organización estructural de los objetos que envían y reciben mensajes.

6. **Diagrama de estados:** muestra una máquina de estado que consiste de estados, transiciones, eventos y actividades.

7. **Diagrama de actividad:** tipo especial de un diagrama de estado que muestra el flujo de una actividad a otra actividad dentro de un sistema.

8. **Diagrama de componentes:** muestra las organizaciones y dependencias entre un conjunto de componentes.

9. **Diagrama de distribución física:** muestra la configuración de los nodos de procesamiento en tiempo de corrida y los componentes que viven en ellos.

Para la representación gráfica de los patrones que se proponen en este trabajo, se usan diagramas de clases que muestran las clases que están involucradas y las relaciones entre ellas; diagramas de secuencia o de interacción que muestran en orden cronológico cómo van sucediendo las operaciones; y diagramas de estados que nos muestran cómo va cambiando de un estado a otro el objeto de la clase principal de cada patrón.

3.2 PATRONES

Los Patrones de Software son una forma literaria diseñada para comunicar conocimiento experto sobre la construcción de un sistema; describen un problema y su solución general en un contexto particular [3].

Los patrones tienen sus orígenes en el diseño urbano y en la arquitectura de construcciones en el trabajo de Christopher Alexander. Se cree que la comunicación humana es un obstáculo en el desarrollo de software; si la forma literaria de un patrón puede ayudar a que los programadores se comuniquen con sus clientes, compañeros de

trabajo y demás personas entonces los patrones ayudan a satisfacer una necesidad crucial del desarrollo contemporáneo de software [3].

Los patrones no son un método completo de diseño ni son una herramienta CASE. Se enfocan más en las actividades humanas de diseño que en las transformaciones que pueden ser automatizadas. Tampoco son inteligencia artificial; realzan y fortalecen la inteligencia humana que separa a las personas de las computadoras [3].

Un patrón es una abstracción porque se enfoca en el problema a un nivel general de forma idónea, aunque la solución puede implicar detalles. La solución tiene suficiente detalle para que el diseñador sepa qué hacer, pero a la vez también es general para expresar un contexto amplio [3].

Los patrones ayudan a expresar experiencia en el desarrollo de software, y promueven las buenas prácticas del diseño. Todos los patrones tratan problemas específicos que suceden en muchas ocasiones en el diseño o implementación de un sistema de software [2].

Algunos patrones ayudan a dividir un sistema en subsistemas, otros a implementar aspectos particulares de diseño en un lenguaje de programación específico. También existen patrones que capturan las mejores prácticas de proceso de desarrollo de software.

Un patrón resuelve un problema particular, pero su aplicación puede desencadenar nuevos problemas que pueden ser resueltos por otros patrones. Los componentes o relaciones únicas dentro de un patrón particular pueden ser descritos por patrones más pequeños que más tarde pueden ser integrados y estar contenidos en un patrón más grande.

Los patrones deben ser presentados de una forma adecuada para que se entiendan, también deben ser descritos uniformemente para que se pueda comparar más fácilmente un patrón con otro. Sin embargo, no hay un estándar de secciones que debe incluir un patrón.

Un patrón debe tener un nombre propio que lo identifique claramente, y usa diagramas y escenarios para ilustrar los aspectos estáticos y dinámicos de la solución.

Existe una gran variedad de formas establecidas de patrones; a continuación se mencionan las más comunes:

1. **Alexanderian:** Surge del trabajo de Christopher Alexander, es la forma original de patrón. Las secciones de un patrón Alexanderian no están muy delimitadas. La estructura sintáctica principal es un **Por lo tanto** inmediatamente precediendo la solución. Otros elementos de la forma están comúnmente presentes como son un enunciado del problema, una discusión de las fuerzas, la solución y lo racional [3].

2. **GOF** (Gang of Four): Esta forma fue establecida en **Design Patterns** [6] y está enfocada hacia los diseños de software orientados a objetos [3].

3. **Portland:** Es una forma muy parecida a la Alexanderian pero más simplificada en su composición [3].

4. **Coplien:** Esta forma también refleja los elementos básicos encontrados en la forma Alexanderian [3].

La forma de patrón que se utiliza en este trabajo es parecida a la forma de Coplien, consta de las siguientes secciones:

- **Nombre del patrón.**
- **Problema,** describe un problema que sucede repetidamente en un contexto dado.
- **Contexto,** muestra las situaciones en las que ocurre el problema.
- **Fuerzas,** discute diferentes puntos de vista del problema y ayuda a entender sus detalles. Cada una de estas fuerzas puede complementar o contradecir a las otras.
- **Solución,** explica cómo resolver el problema y se divide en descripción general, aspectos sociales y una sección orientada a objetos.
- **Contexto Resultante,** describe la modificación del contexto a causa de la aplicación de la solución.
- **Patrones Relacionados,** menciona aquellos patrones que se utilizan para elaborar el patrón actual.
- **Lo Racional,** es una sección opcional que incluye la justificación de las decisiones hechas para definir la solución.

La sección **Solución** a su vez contiene tres partes:

- **Descripción General,** contiene la presentación de la solución en lenguaje natural.
- **Aspectos Sociales,** describe los elementos humanos de la solución que deberían tomarse en cuenta.
- **Orientada a Objetos,** expresa la solución de una manera más técnica y práctica utilizando modelado orientado a objetos.

Las dos primeras partes de la sección **Solución** están dirigidas a los administradores y líderes de los equipos de software quienes deben proporcionar las condiciones para la implementación del patrón.

La tercera parte está dirigida a los ingenieros de software quienes deben implementar el patrón.

Con toda esta información disponible, un desarrollador cuenta con más elementos para entender un determinado patrón, aplicarlo e implementarlo con mayor facilidad.

PARTE 2 GUIA DE MÉTRICAS BÁSICAS DE SOFTWARE

CAPÍTULO 4 INTRODUCCIÓN A LA GUÍA DE MÉTRICAS BÁSICAS DE SOFTWARE

Para elaborar los dos patrones de esta guía se tomaron como base cuatro patrones denominados **Patrones Básicos de Métricas para la Mejora del Proceso de Software**, los cuales fueron elaborados por la Dra. Hanna Oktaba y la Maestra Guadalupe Ibargüengoitia [10]. El primero es el **Patrón de Registro de Tiempo de una Actividad**, el cual consiste en contar y registrar el tiempo que se invierte en las actividades del proceso de software. El segundo es el **Patrón de Registro de Tamaño del Producto**, el cual calcula el tamaño de los productos de software. El tercero es el **Patrón de Registro de Búsqueda de Defectos**, que registra los defectos encontrados en los productos de software. Y el cuarto es el **Patrón de Registro de Corrección de Defectos**, que consiste en registrar la información sobre defectos corregidos en los productos de software.

4.1 OBJETIVO

Proporcionar una guía mediante la cual un ingeniero de software pueda calcular métricas básicas de un producto de software, siguiendo una serie de pasos cortos, bien definidos y explicados, tanto textualmente como gráficamente, utilizando un modelado orientado a objetos, de manera que sea comprensible y entendible para aquellas personas que tengan la necesidad de utilizarla.

4.2 CONTENIDO

En esta guía de métricas básicas de software se proponen dos patrones, el primero es el **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones**, el cual consiste en calcular los tiempos reales que se dedicaron a la producción, pruebas y correcciones de un producto de software desde que se inicia su creación hasta que ingresa

por primera vez bajo el control de configuración. Estas métricas sirven para mejorar el proceso de software, estimar tiempos y mejorar técnicas de prueba en proyectos futuros.

El segundo, es el **Patrón de Indicadores sobre Defectos Encontrados y Corregidos** que consiste en calcular algunos indicadores cuantitativos de los defectos encontrados y corregidos de un producto de software, por ejemplo, los defectos encontrados, los defectos corregidos, los defectos no corregidos, los defectos por tamaño, los defectos encontrados y corregidos, los defectos encontrados por hora de pruebas y los defectos corregidos por hora de correcciones. Estas métricas permiten tener estimaciones más certeras en tiempo, recursos y número de defectos que se van a introducir y a remover en proyectos futuros, lo cual resulta en una mejora en la calidad del producto y en los procesos de producción.

Los patrones **Registro de Tiempo de una Actividad**, **Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos**, se aplican para llevar a cabo el **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones**, ya que en éste las personas que realizan las actividades de producción, prueba y corrección deben registrar el tiempo que dedican a ellas y lo hacen siguiendo dichos patrones.

De la misma manera, el **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones** y el **Patrón de Registro de Tamaño del Producto** proporcionan al **Patrón de Indicadores de Defectos Encontrados y Corregidos** los registros de tiempos y de tamaño del producto para que realice sus cálculos y así obtener las métricas básicas de defectos. También este patrón es complementario a los patrones **Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos** ya que agrega al registro de defectos que resulta de estos patrones, información relacionada con los totales de defectos encontrados y corregidos.

Para que sea autocontenida esta guía, el capítulo 5 contiene la traducción al español de los cuatro patrones básicos de métricas [10] desarrollados por la Doctora Hanna Oktaba y la Maestra Guadalupe Ibargüengoitia. Los capítulos 6 y 7 contienen los patrones desarrollados por la autora de este trabajo.

El propósito de elaborar la **Guía de Métricas Básicas de Software** es que sea de utilidad para cualquier empresa. Hoy en día, sabemos que casi todas las empresas cuentan con un área de desarrollo de sistemas, lo cual significa la existencia de productos de software, considerando como producto cualquier artefacto, entregable o documento que resulte de una actividad del proceso de software. Se deben obtener métricas relacionadas con este producto. Por tal motivo, se podría dar a conocer esta Guía a los ingenieros de software para que la llevaran a la práctica en las diversas actividades de su proceso de software.

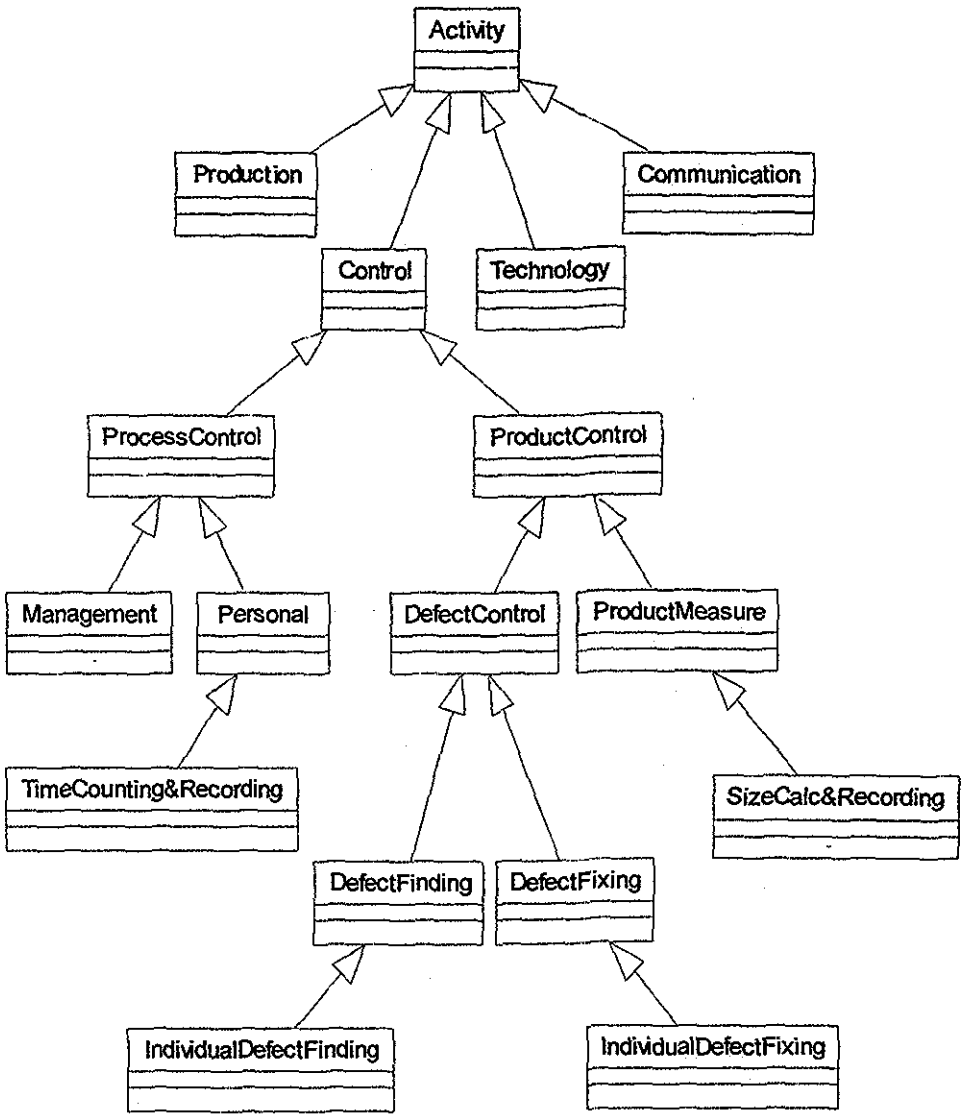
4.3 TAXONOMÍA DE LAS ACTIVIDADES DEL PROCESO DE SOFTWARE

En esta sección se muestra un diagrama de clases de las actividades del proceso de software que fue presentado en [10] y que es básico para comprender los patrones que en capítulos posteriores se desarrollan. Las actividades del proceso de software se clasifican en cuatro grupos: producción, control, tecnología y comunicación.

Las actividades de producción son las que se relacionan con la construcción de un producto de software. Las de control son las que verifican el estado del proceso y de los productos que generaron las otras actividades. Las de tecnología son la evaluación del software y hardware, reuso, etc. Y las de comunicación son las que se llevan a cabo entre los roles.

Las actividades de control se especializan en dos tipos: control del proceso y control del producto.

Las actividades de control del proceso se especializan en dos actividades, una relacionada con actividades de administración y otra con actividades personales. Las actividades de control del producto son las que se dedican a encontrar y corregir defectos. También, control del producto se especializa en medición del producto, cuyas actividades llevan a cabo el cálculo de tamaño y su registro.



Taxonomía de las actividades del Proceso de Software

CAPÍTULO 5 PATRONES BÁSICOS DE MÉTRICAS PARA LA MEJORA DEL PROCESO DE SOFTWARE [10]

Los patrones básicos de métricas para la mejora del proceso de software son la base para la realización de los dos patrones que se proponen en esta guía; éstos toman los tiempos recolectados por aquellos, el tamaño del producto e información sobre los defectos para obtener el tiempo total invertido en cada una de las actividades del proceso de software y calcular algunos indicadores sobre defectos encontrados y corregidos.

Los patrones básicos de métricas para la mejora del proceso de software son:

1. **Patrón de Registro de Tiempo de una Actividad:** Registra el tiempo que se invierte en las actividades del proceso de software.

2. **Patrón de Registro de Tamaño del Producto:** Calcula el tamaño de los productos de software.

3. **Patrón de Registro de Búsqueda de Defectos:** Registra los defectos encontrados en los productos de software.

4. **Patrón de Registro de Corrección de Defectos:** Registra la información sobre defectos corregidos en los productos de software.

A continuación se describen ampliamente cada uno de ellos.

5.1 PATRÓN DE REGISTRO DE TIEMPO DE UNA ACTIVIDAD

Problema

Es necesario contar y registrar el tiempo que se invierte en las actividades del proceso de software [10].

Contexto

Se desarrollan las actividades del proceso de software sin poner mucha atención en conocer el tiempo exacto que se invierte en ellas. Se reporta al líder o administrador datos aproximados en términos de horas, días o semanas. Sin embargo, para evaluar cuantitativamente el costo del proyecto de software y la productividad del equipo de desarrollo de software, es necesario saber el tiempo efectivo que los miembros del equipo invierten en las actividades definidas en el proceso de software. El tiempo efectivo significa, el tiempo que invertimos entre el inicio de la actividad y su terminación menos el tiempo perdido durante las diferentes interrupciones (llamadas telefónicas, descansos, pláticas con colegas, etc.) las cuales causan la suspensión temporal de la actividad. El tiempo que actualmente se dedica a la ejecución de la actividad es el tiempo que se quiere registrar como tiempo efectivo [10].

Fuerzas

La recolección de datos de tiempo efectivo requiere disciplina y honestidad. Los miembros del equipo intentarán excluir las interrupciones porque esto hace evidente el tiempo actual que dedican al trabajo. Por otro lado, los administradores necesitan datos confiables de tiempo para calcular los costos y productividad de los proyectos [10].

Solución

Descripción General

Se inicia la actividad con algún tipo de inicialización (se preparan las entradas y las herramientas). Luego se empieza su ejecución y se registra la fecha y hora de inicio. Durante la ejecución de la actividad se puede suspenderla y después continuar en repetidas ocasiones, registrando el período de cada interrupción. Finalmente, la actividad se termina cuando produce el resultado esperado (producto) y se calcula el tiempo efectivo que se ha invertido en ella. En ocasiones se abandona (aborta) la actividad por diversas razones pero el tiempo invertido tiene que ser registrado y contado, aunque sea probablemente tiempo perdido [10].

Aspectos Sociales

Para disminuir la resistencia de los ingenieros de software para recolectar los datos de tiempo, los administradores deben [10]:

1. Definir la política organizacional, que explica el propósito de la recolección de datos de tiempo y excluye el uso de esos datos en la evaluación individual de los miembros del equipo.

2. Proporcionar herramientas automáticas de soporte.

3. Ofrecer capacitación en el proceso y las herramientas.

Especificación Orientada a Objetos

El diagrama de transición de la ***Fig. 1*** muestra los estados posibles de los objetos de la clase **Activity** que representa las actividades del proceso de software [10].

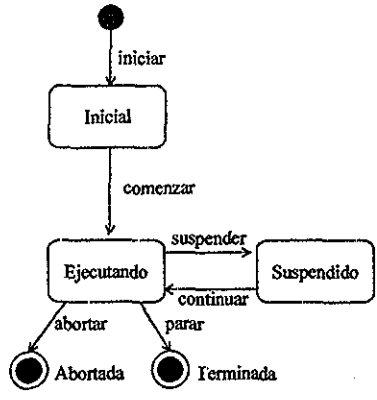


Fig. 1. Diagrama de transición de estados de la clase Activity.

Se expresan los cambios de los estados a través del atributo **status** y los eventos que causan estos cambios como los métodos de la clase **Activity** (***Fig. 2***). Para registrar el tiempo efectivo de ejecución de la actividad se le asocia con otra de control de proceso personal que corresponda con el conteo y registro de tiempo (ver ***Fig. 2***) [10].

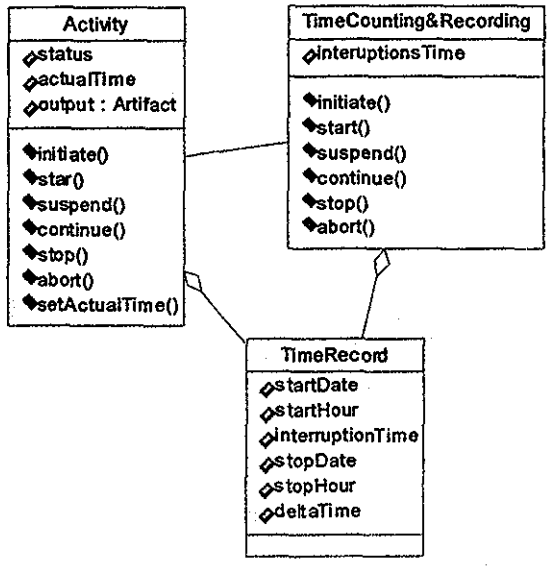


***Fig. 2.** Diagrama de clases del patrón Registro de Tiempo de una Actividad.*

Los métodos de **Activity** pueden ser expresados como sigue [10]:

- **initiate** – inicia el estado de la actividad, prepara las entradas y las herramientas e inicia la actividad de conteo y registro de tiempo,
- **start** – cambia el estado a "Ejecutando", empieza el conteo y registro de tiempo y ejecuta la actividad "Principal",
- **suspend** -- suspende la actividad, cambia el estado a "Suspendido" y suspende el conteo y registro de tiempo,
- **continue** -- cambia el estado a "Ejecutando", continúa con el conteo y registro de tiempo y con la actividad misma,
- **stop** – termina la actividad, cambia el estado a "Terminado" y se detiene el conteo y registro de tiempo,
- **abort** – termina la actividad, cambia el estado a "Abortado" y se detiene el conteo y registro de tiempo.

La clase **TimeCounting&Recording** hereda los métodos de **Activity** redefinidos como sigue [10]:

- **initiate** – inicia el **interruptionsTime** en 0,
- **start** – registra la fecha y hora actual,
- **suspend** – registra la fecha y hora de inicio de la interrupción,
- **continue** – registra la fecha y hora de fin de la interrupción y computa el tiempo acumulado de interrupción,
- **stop** – registra la fecha y hora de paro, computa el tiempo efectivo de la actividad (deltaTime), crea el registro de tiempo final (objeto de la clase **TimeRecord**) y lo asigna a la actividad terminada como el valor de **actualTime**,
- **abort** – registra de la misma manera que en el método **stop**; el estado final de la actividad puede ayudar a decidir cómo usar la información de registro de tiempo en este caso.

La ***Fig. 2*** muestra el diagrama de clases que expresa la vista estática de la asociación entre cualquier actividad y la actividad de conteo y registro de tiempo. La clase **TimeRecord** es una especialización de la clase abstracta **Artifact** y es una agregación de la clase **Activity**. La ***Fig. 3*** muestra el diagrama de interacción que expresa la secuencia temporal de las invocaciones de los métodos. Observe que la ejecución de los métodos de la clase **Activity** es hecha por humanos mientras la ejecución de los métodos de **TimeCounting&Recording** puede ser hecha por humanos así como por una herramienta automática [10].

Nota: este patrón no se aplica a la actividad de conteo y registro de tiempo, debido a la invocación recursiva de los métodos.

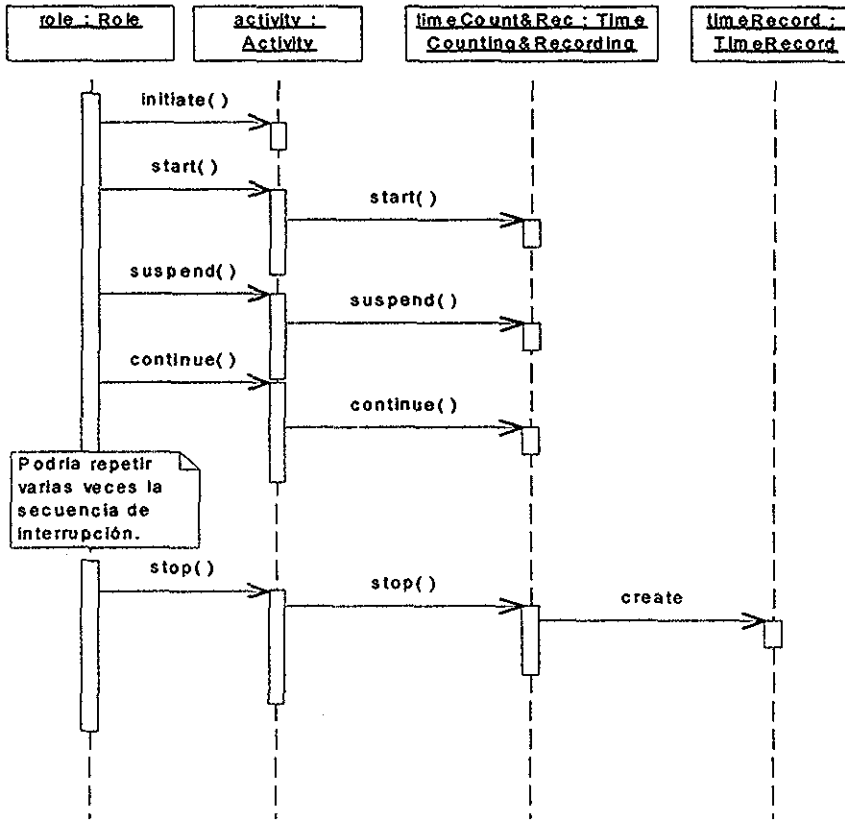


Fig. 3. Diagrama de interacción del patrón Registro de Tiempo de una Actividad

Contexto Resultante

Se mide el tiempo efectivo dedicado a la ejecución de las actividades del proceso de software [10].

Patrones Relacionados

Para obtener la productividad del proyecto se necesitan los datos de tiempo y tamaño. Así, es conveniente combinar este patrón con el patrón **Registro de Tamaño del Producto** en el proceso de desarrollo de software [10].

5.2 PATRÓN DE REGISTRO DE TAMAÑO DEL PRODUCTO

Problema

Es necesario calcular el tamaño de los productos de software [10].

Contexto

Para cuantificar la productividad es necesario conocer el tamaño de los productos de software generados por la ejecución de la actividad del proceso de software. No hay una medida única para los productos de software. Probablemente, el número de líneas de código (LOC) y los puntos de función son las métricas más populares de tamaño de código. Con respecto a las métricas para los productos de análisis y diseño, la situación es aún peor. El problema básico con la selección de las métricas de tamaño para los productos de software es que deben ser aplicables de manera consistente, precisa y automatizada. Las definiciones de métricas de software también deben considerar los aspectos de reuso [10].

Otro problema es elegir correctamente el momento para evaluar el tamaño del producto de software. Se sabe que los productos cambian constantemente en la corrección de defectos, mejoras y extensiones.

Fuerzas

Los miembros del equipo se resistirán a los cálculos de tamaño si no hay soporte automatizado para ello. También, aumentarán sus datos de tamaño para tener una mejor productividad. Por otro lado, los administradores necesitan los datos confiables de tamaño para calcular los costos y la productividad real de los proyectos [10].

Solución

Descripción General

Se elige la métrica de tamaño consistente y se precisa para cada uno de los productos de software generados por las actividades de producción. Se define el tipo de unidades que *se contarán, por ejemplo LOC, puntos de función u otro. Se escoge o desarrolla* las herramientas para apoyar el conteo. Luego se registra el tamaño del producto de software terminando cada actividad, la cual genera una nueva versión del producto. Asocia el registro de tamaño a aquella versión del producto [10].

Aspectos Sociales

Para disminuir la resistencia del ingeniero de software para recolectar los datos de tamaño, los administradores deben [10]:

1. Definir la política organizacional, la cual explica el propósito de la recolección de datos de tamaño y excluye el uso de estos en la evaluación individual de los miembros del equipo.

2. Proporcionar herramientas automáticas para apoyarlo.

3. Proporcionar el entrenamiento y las herramientas para llevar a cabo el patrón.

Especificación Orientada a Objetos

Se asocia la clase **Activity** a la clase **SizeCalc&Recording** (*Fig. 4*) la cual representa la especialización de la actividad de control del producto. Esta clase recibe como entrada el producto de salida de la actividad "principal" y ofrece el servicio de cálculo y registro de tamaño (**sizeCalc&Rec( )**). El producto de salida de la actividad de cálculo y registro de tamaño está representado por la clase **SizeRecord** que recolecta el número de unidades. El algoritmo de cálculo depende del tipo de unidades de tamaño, definidas por el producto (el atributo **unitType** de la clase **Artifact**) y puede ser expresado como una sobrecarga del método **sizeCalc&Rec( )**. Se registra el tamaño del producto terminando la ejecución de la actividad. Para expresarlo en el modelo orientado a objetos se tiene que incluir las siguientes acciones en el método **stop** de la clase **Activity** [10]:

- **stop** – inicia la actividad de cálculo y registro de tamaño y transfiere a ésta como un parámetro de entrada, el producto de salida (producto de software) de la actividad "principal", luego calcula y registra el tamaño del producto (número de unidades), finalmente, asigna el registro de tamaño al producto.

La ***Fig. 4*** muestra las dependencias estáticas de las clases involucradas en este patrón, y la ***Fig. 5*** presenta la cooperación dinámica en un diagrama de interacción.

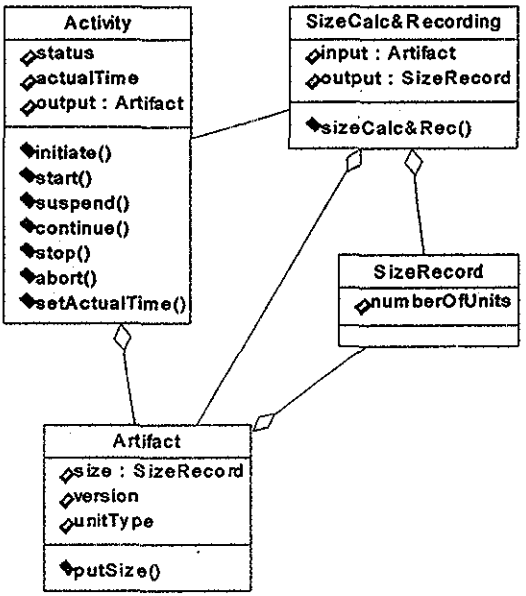


Fig. 4. *Diagrama de clases del patrón Registro de Tamaño del Producto*

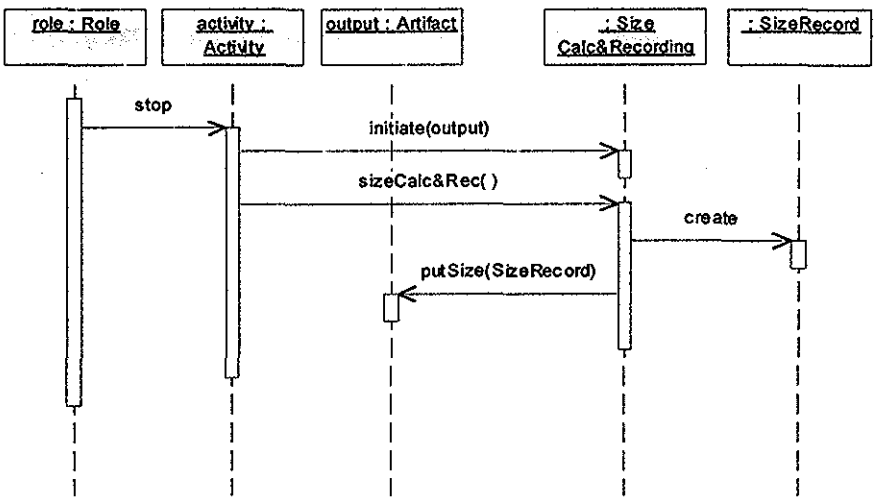


Fig. 5. *Diagrama de interacción del patrón Registro de Tamaño del Producto*

Contexto Resultante

Se conoce el tamaño de cada versión de los productos generados por las actividades de producción o corrección de defectos [10].

Patrones Relacionados

Para obtener la productividad del proyecto son necesarios los datos de tamaño y tiempo. Por tal motivo, es conveniente combinar este patrón con el patrón **Registro de Tiempo de una Actividad** en el proceso de desarrollo de software.

Este patrón no calcula la diferencia de tamaño entre dos versiones del producto, la cual se puede definir en términos de las unidades agregadas, modificadas, reutilizadas y borradas. La diferencia de tamaño es importante para evaluar el radio preciso de productividad. El patrón **Registro de Corrección de Defectos** muestra cómo se puede incluir la diferencia de tamaño como una medida adicional asociada a las actividades, que cambia los productos [10].

5.3 PATRÓN REGISTRO DE BÚSQUEDA DE DEFECTOS

Problema

Es necesario registrar los defectos encontrados en los productos de software [10].

Contexto

La calidad de los productos de software está fuertemente relacionada a la habilidad de buscar y corregir defectos. Las actividades básicas del proceso de software dedicadas para encontrar defectos en los productos de software son: revisiones, inspecciones, compilación y prueba. Se desea conocer la efectividad de las técnicas aplicadas en estas actividades y mejorarlas. Es necesario reunir los datos acerca de los defectos para que se puedan analizar y corregir [10].

Fuerzas

Hay una resistencia natural de los humanos para mostrar a otros nuestras fallas. Es más fácil buscar defectos en el trabajo de alguien más. Por otro lado, para mejorar el proceso de software y la calidad de los productos de software se necesita aprender de nuestros errores [10].

Solución

Descripción General

La actividad de búsqueda de defectos, tal como la revisión, inspección, compilación o prueba, podría ser vista como una secuencia de actividades individuales de búsqueda de defectos que terminan cuando encuentran algunos defectos en un producto. Para realizar la actividad individual de búsqueda de defectos se llevan a cabo los siguientes pasos [10]:

1. Cuando empieza la actividad individual de búsqueda de defectos se inicia el conteo de tiempo que se invertirá en ella, utilizando el patrón **Registro de Tiempo de una Actividad**.

2. Cuando se encuentra un defecto se crea un registro que contenga la siguiente información: tiempo que se invirtió en buscarlo, tipo de defecto, fase probable del proceso en el que fue introducido, fase del proceso en el que fue encontrado, descripción del defecto y el estatus de defecto no corregido.

3. Incluir el registro de defectos en el log de registro de defectos del producto.

Aspectos Sociales

Para disminuir la resistencia del ingeniero de software para recolectar los datos de defectos, los administradores deben [10]:

1. Definir la política organizacional, la cual explica el propósito de la recolección de datos de defectos y excluir el uso de estos datos en la evaluación individual de los miembros del equipo.

2. Proporcionar herramientas automáticas para apoyar esta política.

3. Promover los métodos de búsqueda de defectos (inspecciones y pruebas) ejecutados por las personas, quienes no son los autores de los productos examinados.

4. Proporcionar el entrenamiento y las herramientas para llevar a cabo este patrón.

Especificación Orientada a Objetos

Cualquier actividad de búsqueda de defectos podría ser vista como una secuencia de actividades individuales de búsqueda de defectos. La clase **IndividualDefectFinding** es una subclase indirecta de la clase **Activity** ya que hereda todos sus atributos y métodos. La búsqueda individual de defectos inicia tomando como un artefacto de entrada al producto de software que es el objeto de búsqueda de defectos. Se inicia la ejecución de la actividad individual de búsqueda de defectos y se cuenta el tiempo invertido en ella aplicando el patrón **Registro de Tiempo de una Actividad**. Cuando se encuentra un defecto se termina la actividad redefiniendo el método **stop** como sigue [10]:

- Invocar el método **stop** de la superclase que genera el registro de tiempo actual (ver el patrón **Registro de Tiempo de una Actividad**).

- Crear el registro de defectos (objeto de la clase **DefectRecord**) con los siguientes datos: tiempo invertido en buscar el defecto (registro de tiempo actual), tipo de defecto, fase del proceso en la que el defecto fue encontrado, fase del proceso en la que el defecto fue introducido, descripción del defecto y un estatus de no corregido.

- Insertar el registro al log de registro de defectos del producto de software (artefacto).

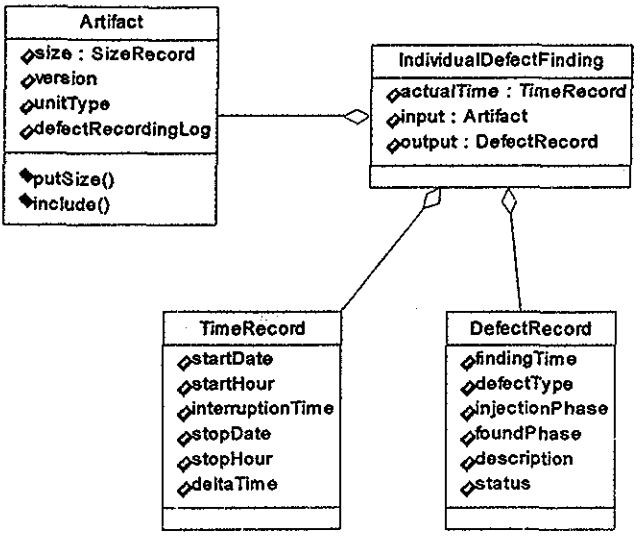


Fig. 6. Diagrama de clases del patrón Registro de Búsqueda de Defectos.

La ***Fig. 6*** representa la relación estática entre las clases involucradas en este patrón y la ***Fig. 7*** muestra el diagrama de interacción para sus objetos.

Se repite la actividad individual de búsqueda de defectos hasta que se desee y se obtiene información acumulada de defectos almacenada en el **defectRecordingLog** del producto [10].

Contexto Resultante

Se tiene el registro de los defectos encontrados en los productos de software examinados por el método de búsqueda de defectos. Este registro puede ser utilizado como una guía para la actividad de corrección de defectos [10].

Patrones Relacionados

Un patrón involucrado en la descripción de este patrón es el de **Registro de Tiempo de una Actividad** utilizado para conocer el tiempo invertido para buscar los defectos. El patrón complementario al de **Registro de Búsqueda de Defectos** es el de **Registro de Corrección de Defectos**, que agrega al registro de defectos la información relacionada a la actividad de corrección de defectos [10].

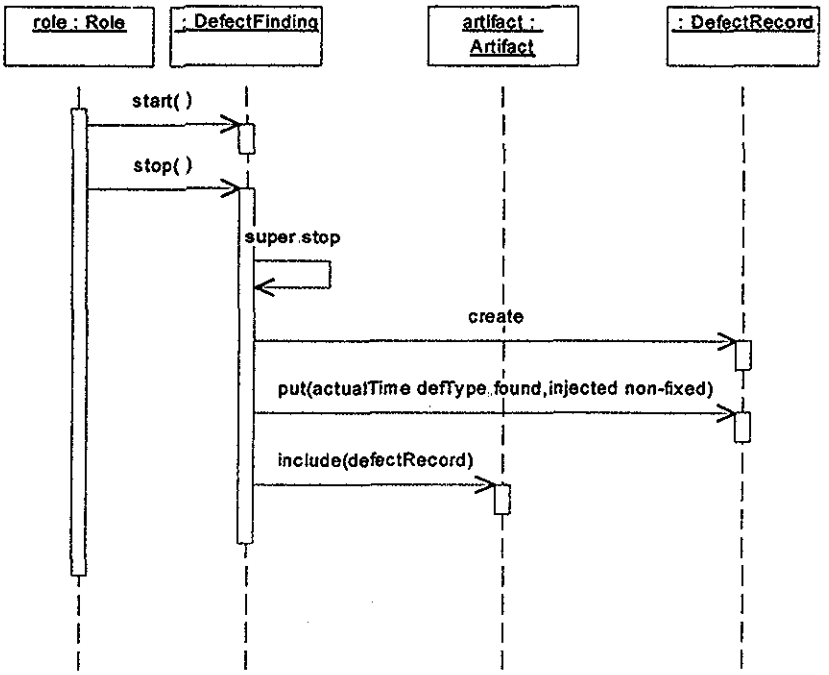


Fig. 7. Diagrama de interacción del patrón Registro de Búsqueda de Defectos.

5.4 PATRÓN DE REGISTRO DE CORRECCIÓN DE DEFECTOS

Problema

Es necesario registrar la información sobre defectos corregidos en los productos de software [10].

Contexto

La actividad de corrección de defectos del proceso de software es comúnmente conocida como "debugging". La fase de mantenimiento y sus actividades correctivas, adaptativas y perfectivas también podrían ser consideradas como de corrección de defectos. Aún si se hacen algunas extensiones del sistema, se está haciendo corrección de defectos que el cliente detecta. Es importante tener el registro histórico de los defectos corregidos en un producto de software. Puede ayudar a modificaciones futuras y permite regresar a

versiones previas. También es útil tener el registro de tiempo de las actividades de corrección de defectos y la evaluación de tamaño de los cambios [10].

Fuerzas

Es difícil tener disciplina para anotar cada defecto corregido. Por otro lado, se necesita información exacta sobre los cambios que se realizan en los productos de software, el tiempo que se invierte en las actividades de corrección de defectos y el tamaño de las modificaciones. Esta información puede ser utilizada por administradores para registrar las actividades de corrección de defectos y estimar la calidad del producto [10].

Solución

Descripción General

La actividad de corrección de defectos podría ser vista como una secuencia de actividades individuales de corrección de defectos, las cuales inician con un análisis de la descripción de defectos del producto y termina cuando el defecto es corregido. Para realizar la actividad de corrección de defectos sigue estos pasos [10]:

1. Cuando empieza la actividad individual de corrección de defectos inicia el conteo del tiempo que se invertirá en ella, utilizando el patrón **Registro de Tiempo de una Actividad**.

2. Se elige el registro de defectos no corregidos asociado al producto.

3. Se analiza la descripción del defecto y se corrige.

4. Cuando se termine, se completa el registro de defectos con la siguiente información: tiempo que se invierte en corregirlo, descripción del cambio, fase del proceso en el que el defecto fue removido, el número de unidades cambiadas (agregadas, modificadas y borradas), y el estatus de defecto corregido.

Se repite la actividad de corrección de defectos seleccionando los siguientes defectos no corregidos hasta que se corrijan todos ellos.

Aspectos Sociales

Como complemento a los aspectos sociales 1, 2 y 4 del patrón **Registro de Búsqueda de Defectos** que se aplica también en este patrón, se puede incluir lo siguiente [10]:

Es importante analizar los datos recolectados en los logs de registros de defectos y usar los resultados para mejorar el proceso de desarrollo de software y no manipularlos para evaluar a los miembros del equipo.

Especificación Orientada a Objetos

Cualquier actividad de corrección de defectos podría ser vista como una secuencia de actividades individuales de corrección de defectos. La clase **IndividualDefectFixing** es una subclase indirecta de la clase **Activity** ya que hereda todos sus atributos y métodos. La corrección individual de defectos inicia tomando como entrada el producto de software, el cual es el objeto de la búsqueda de defectos. Se inicia la ejecución seleccionando un registro de defectos no corregido del log de registros de defectos del producto y se empieza a contar el tiempo invertido en ella aplicando el patrón **Registro de Tiempo de una Actividad**. Se analiza y se corrige el defecto. Se termina la actividad redefiniendo el método **stop** como sigue [10]:

- Se invoca el método **stop** de la superclase que genera el registro de tiempo actual (ver el patrón 3).
- Se completar el registro de defectos con los siguientes datos: tiempo invertido en analizar y corregir los defectos (registro de tiempo actual), descripción de los cambios, fase del proceso en la que el defecto fue corregido, el número de unidades agregadas, modificadas y borradas y el estatus corregido.

La ***Fig. 8*** representa la relación estática entre las clases involucradas en este patrón y la ***Fig. 9*** muestra el diagrama de interacción de sus objetos.

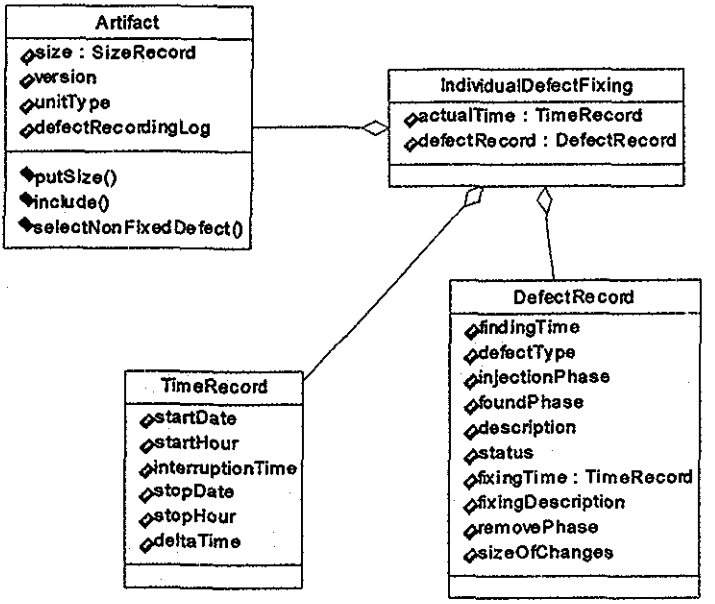


Fig. 8. Diagrama de clases del patrón Registro de Corrección de Defectos.

Se repite la actividad individual de corrección de defectos hasta que se corrijan todos los defectos del producto y se obtiene información completa de los defectos removidos, la cual se encuentra almacenada en el log de registros de defectos del producto.

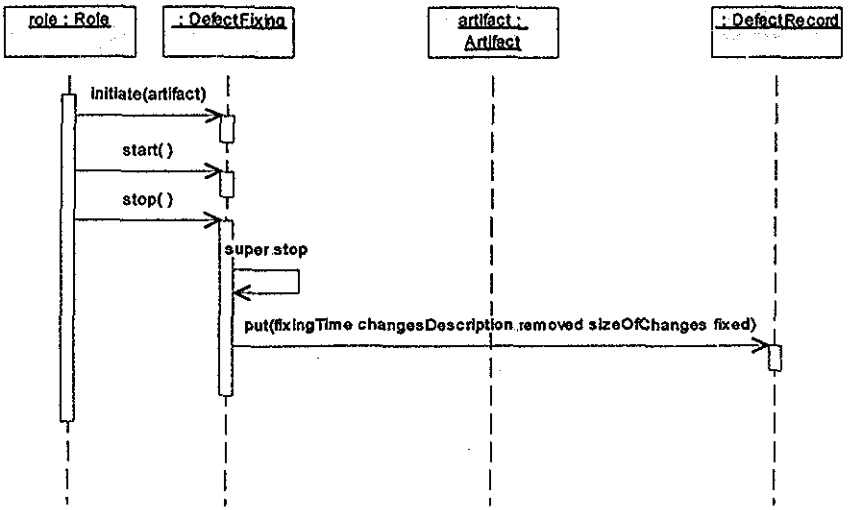


Fig. 9. Diagrama de interacción del patrón Registro de Corrección de Defectos.

Contexto Resultante

Se tiene la información documentada sobre los defectos encontrados y corregidos de un producto de software [10].

Patrones Relacionados

Los patrones involucrados en la presentación de este patrón son: **Registro de Tiempo de una Actividad** y **Registro de Búsqueda de Defectos**. Se usa el primero para conocer el tiempo que se invierte en corregir los defectos y el segundo ayuda a tener un registro de defectos que deben ser removidos.

El patrón **Documentar el Problema**, incluido en el lenguaje del patrón **Prueba del sistema** [4], establece el problema sobre cómo los defectos que fueron encontrados en la prueba deben ser comunicados. La solución dada aquí es simple: "escribir un reporte del problema". Las fuerzas y los aspectos sociales de su solución son similares a los de estos patrones. Pero en la solución de los patrones **Registro de Búsqueda y Corrección de Defectos** pueden encontrarse más detalles sobre qué reportar y cuándo hacerlo [10].

Racional

Se clasifican las actividades de búsqueda y corrección de defectos como las actividades de control del producto diferente a la clasificación de Humphrey [7] como el control del proceso. En nuestra opinión su clasificación es correcta cuando es aplicada al análisis de la información del registro de defectos que conlleva a la mejora de un método particular de búsqueda y corrección de defectos. La colección simple de los registros de defectos, sugerida en PSP y reflejada en estos patrones, es la actividad que permite un mejor control del producto [10].

CAPÍTULO 6 PATRÓN DE REGISTRO DE TIEMPOS DE PRODUCCIÓN, PRUEBAS Y CORRECCIONES.

Resumen

Este patrón calcula los tiempos reales dedicados a la producción, pruebas y correcciones de un producto de software desde su creación inicial hasta su primer ingreso bajo el control de configuración. Utiliza los patrones **Registro de Tiempo de una Actividad, Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos,** toma los tiempos recolectados por estos patrones para realizar una sumatoria y así obtener los tiempos totales.

Problema

Es necesario conocer los tiempos reales dedicados a la producción, pruebas y correcciones de un producto de software desde su creación inicial hasta su primer ingreso bajo el control de configuración, porque con estos datos se calculan la productividad del equipo de trabajo y se realizan estimaciones más exactas del tiempo que se invertirá en un proyecto futuro similar.

Contexto

Se llevan a cabo las actividades de producción, pruebas y correcciones pero no se sabe el tiempo total invertido en cada una de ellas. Eventualmente existe el registro de tiempo total que cada persona invierte en las actividades que realiza pero no hay un cálculo del tiempo invertido por separado en la producción, en pruebas o en correcciones. Este cálculo permite evaluar cuantitativamente la productividad del equipo de desarrollo de software para proponer mejoras en las técnicas de producción, pruebas y correcciones.

Fuerzas

- Es difícil separar los tres tiempos (producción, pruebas y correcciones) dado que el propio autor del producto y otros colegas hacen correcciones a defectos sin realizar registro alguno.

- Se conocen los tiempos (producción, pruebas y correcciones) por separado, puede ayudar a proponer mejoras en técnicas de producción y pruebas, y evaluar cuantitativamente si tuvieron efecto.

Solución

Descripción general

Durante el proceso de desarrollo de un producto de software, desde su creación hasta que ingresa al control de configuración, hay que identificar de manera explícita qué actividades de producción, pruebas y correcciones se llevan a cabo.

Las personas que realizan las actividades de producción, prueba y corrección deben de registrar el tiempo dedicado a ellas siguiendo los patrones previamente definidos para tal fin. Por ejemplo, para las actividades de producción se aplica el patrón **Registro de Tiempo de una Actividad**, para las de prueba se aplica el patrón **Registro de Tiempo de Búsqueda de un Defecto** y para las de correcciones se aplica el patrón **Registro de Tiempo de Corrección de un Defecto**.

En el momento en que el producto quede aprobado para ingresar bajo el control de configuración, se deben sumar los tiempos que todos los miembros dedicaron a las actividades de producción (t_{pro}), pruebas (t_{pru}) y correcciones (t_{cor}). Estos valores se asocian con el producto, y de esta manera queda registrado el tiempo total dedicado a cada una de las actividades de producción, prueba y correcciones. También se deben sumar t_{pro}, t_{pru} y t_{cor} para obtener el tiempo total (t_{total}). Este valor igual que los anteriores, se asocia con el producto para tener registrado el tiempo total dedicado desde la producción hasta la corrección de un producto.

Aspectos Sociales

Los tiempos recolectados en cualquier actividad no deben de utilizarse para evaluar el trabajo individual de las personas.

Es recomendable que todas las personas participantes sigan las mismas técnicas definidas para cada una de las actividades de producción, prueba y corrección.

Orientado a Objetos

Para modelar las actividades de producción, prueba y corrección, se utiliza un diagrama de clases (***Fig. 11***) en el cual las actividades se ubican dentro del proceso de software [10].

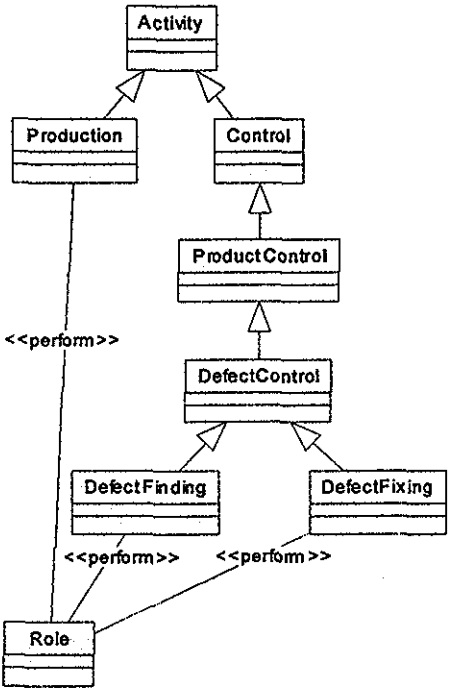


Fig. 11. Modelado de las Actividades de Producción, Prueba y Corrección

En el diagrama, la clase **Activity** (Actividad) se especializa en las clases **Production** (Producción) y **Control** (Control), ésta última hereda a la clase **ProductControl** (ControlProducto) que a su vez hereda a la clase **DefectControl** (ControlDefecto), la cual absorbe a las actividades de búsqueda y corrección de defectos representadas por las clases **DefectFinding** (BúsquedaDefecto) y **DefectFixing** (CorrecciónDefecto).

Las personas que realizan las actividades de producción, prueba y corrección se van a representar por la clase **Role** y deben registrar el tiempo dedicado a ellas aplicando los patrones **Registro de Tiempo de una Actividad, Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos** respectivamente.

La clase **Artifact** (Artefacto) representa al producto de software, un objeto de tipo **Artifact** entra al estado **Production/Fixing/Finding** (Producción/Corrección/Búsqueda) y se registra el tiempo invertido en cada actividad, al salir de este estado, el objeto de tipo **Artifact** es aprobado para ingresar bajo el control de configuración y su estado se vuelve **"Base-lined/Approved"** (Aprobado). El diagrama de estados que refleja este cambio es el de la ***Fig 12***:

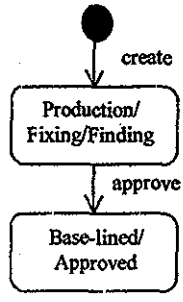


Fig. 12. Diagrama de Estados de la Clase Artifact (Artefacto)

Una vez que el producto se encuentra en el estado **"Approved"** (Aprobado), (1) se deben sumar los tiempos que todos los miembros dedicaron a cada una de las actividades de producción, prueba y correcciones y (2) se suman los tiempos totales de producción, prueba y correcciones. El primer cálculo lo realizarán las clases **TotalProductionTime** (TiempoTotalProducción) para las de producción, **TotalFindingTime** (TiempoTotalBúsqueda) para las de pruebas y **TotalFixingTime** (TiempoTotalCorrección) para las de corrección. El segundo cálculo lo realizará la clase **ArtifactTimeRegisters** (RegistrosTiempoArtefacto). Estos valores acumulados en el objeto de la clase **ArtifactTimeRegisters** se agregan al producto, representado por la clase **Artifact**. Ver ***Fig. 13***.

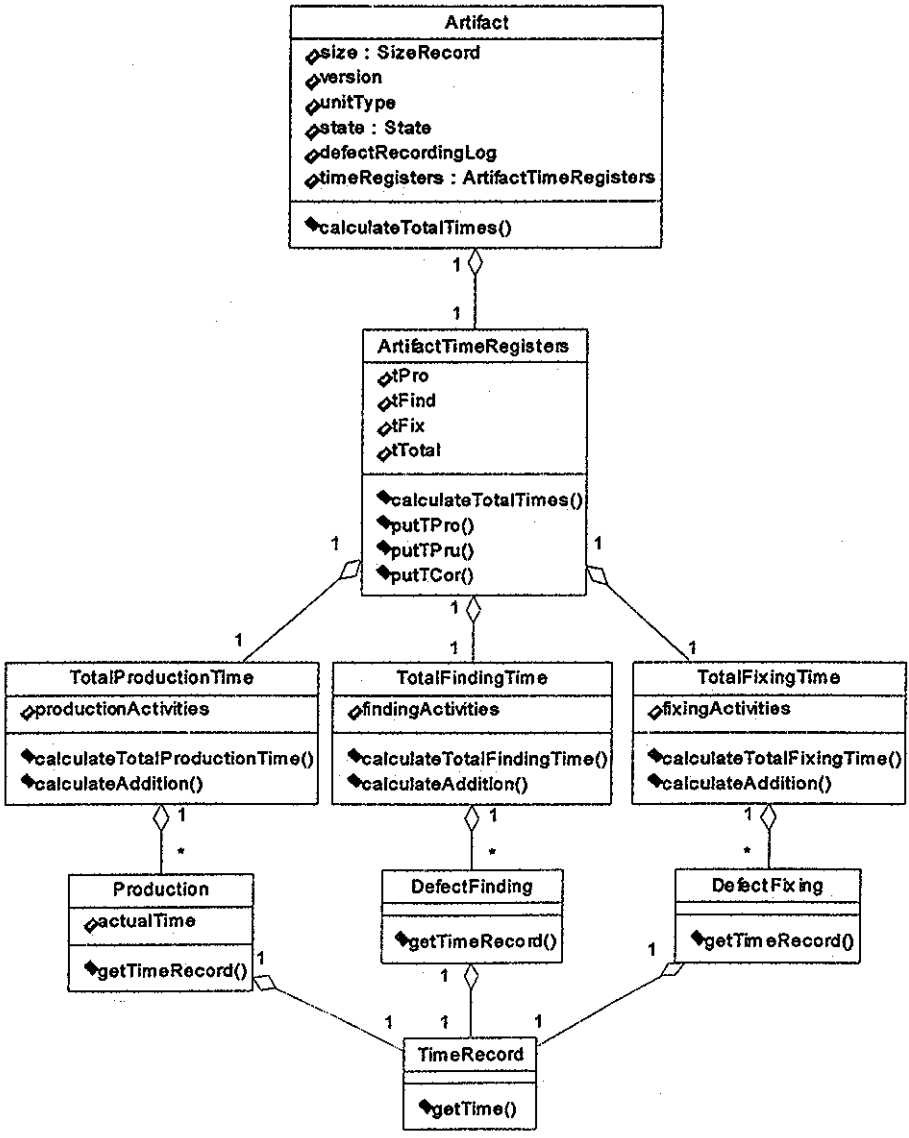


Fig.13. *Diagrama de Clases del Patrón Registro de Tiempos de Producción, Pruebas y Correcciones.*

Los registros de tiempos **TimeRecord** (RegistroTiempo), **Production** (Producción), **DefectFinding** (BúsquedaDefecto) y **DefectFixing** (CorrecciónDefecto) se recolectaron siguiendo los patrones:

- **Activity Time Record** (Registro de Tiempo de una Actividad).
- **Defect Finding Record** (Registro de Búsqueda de Defectos).
- **Defect Fixing Record** (Registro de Corrección de Defectos).

A continuación se muestra el diagrama de interacción que explica de forma detallada cómo se hace la recolección de los tiempos de producción, pruebas y correcciones.

El objeto de tipo **Role** pide al objeto de tipo **Artifact** (Artefacto) que calcule los tiempos totales de producción, pruebas y correcciones; éste a su vez, se lo solicita al objeto **atr** de tipo **ArtifactTimeRegisters** (RegistrosTiempoArtefacto) y éste envía el mismo mensaje a los objetos de tipo **TotalProductionTime** (TiempoTotalProducción), **TotalFindingTime** (TiempoTotalBúsqueda) y **TotalFixingTime** (TiempoTotalCorrección) respectivamente, para que realicen el cálculo.

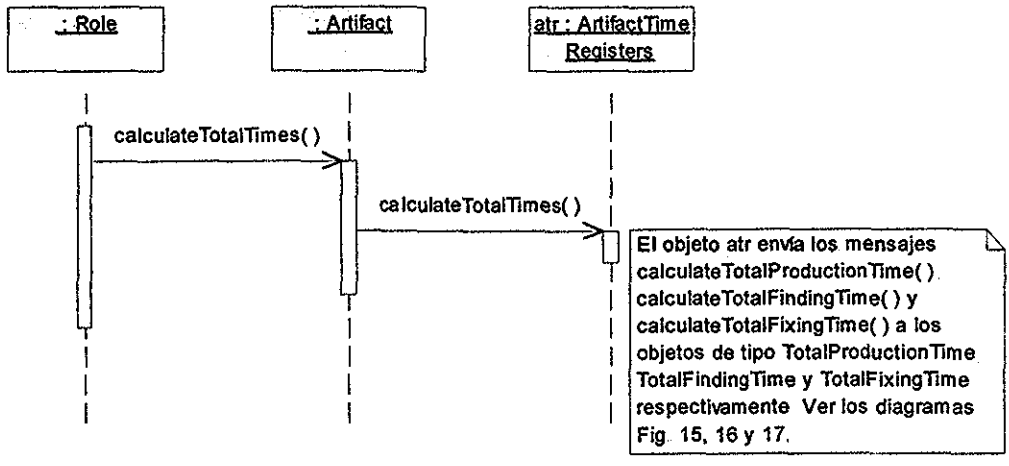


Fig.14. Solicitud de Cálculo de los Tiempos Totales de Producción, Pruebas y Correcciones.

El mensaje que **atr** envía a cada una de las clases agrupadoras (**TotalProductionTime, TotalFindingTime** y **TotalFixingTime**) se presenta en tres diagramas diferentes para que se visualice mejor la secuencia de operaciones de este proceso.

Cada una de las actividades (producción, pruebas y correcciones) al terminarse, tiene incluido el registro de tiempo. El objetivo de crear las clases agrupadoras es para que calculen el tiempo total, es decir, realicen la sumatoria y luego registren el tiempo total en la clase **ArtifactTimeRegisters** (RegistrosTiempoArtefacto).

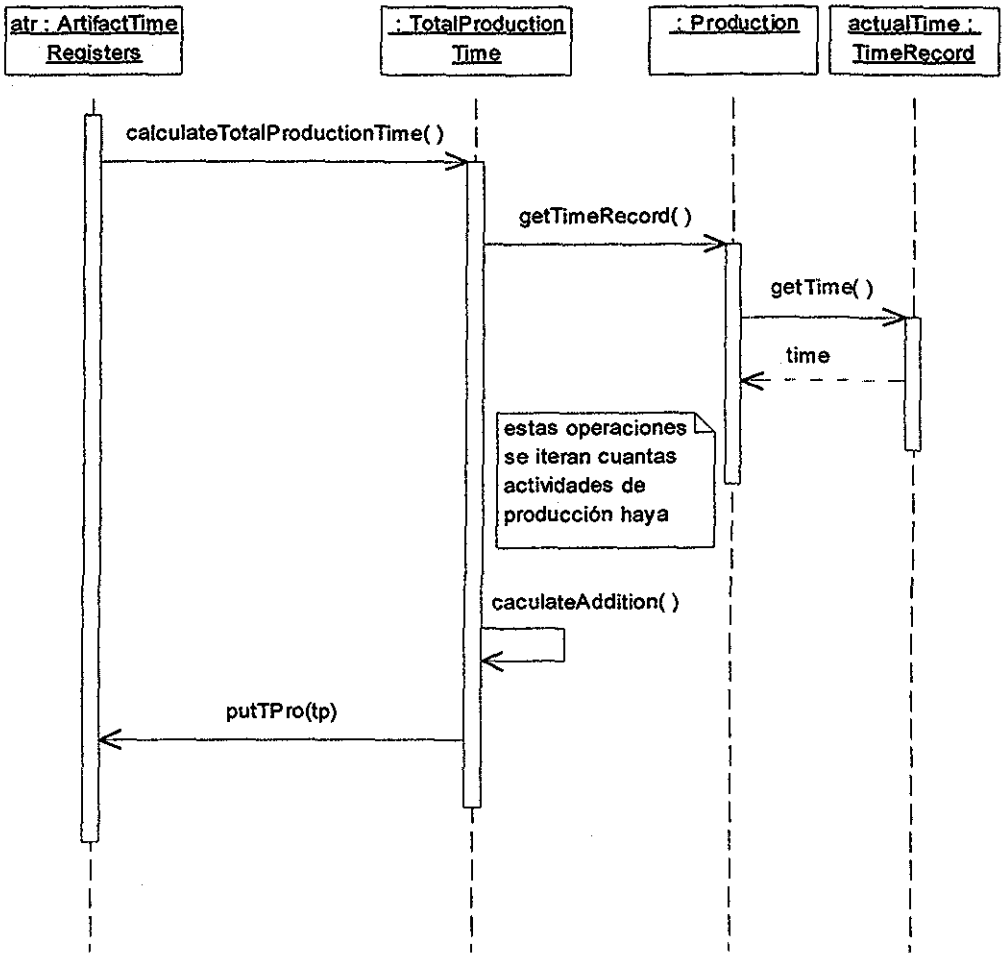


Fig. 15. Tiempo total de producción

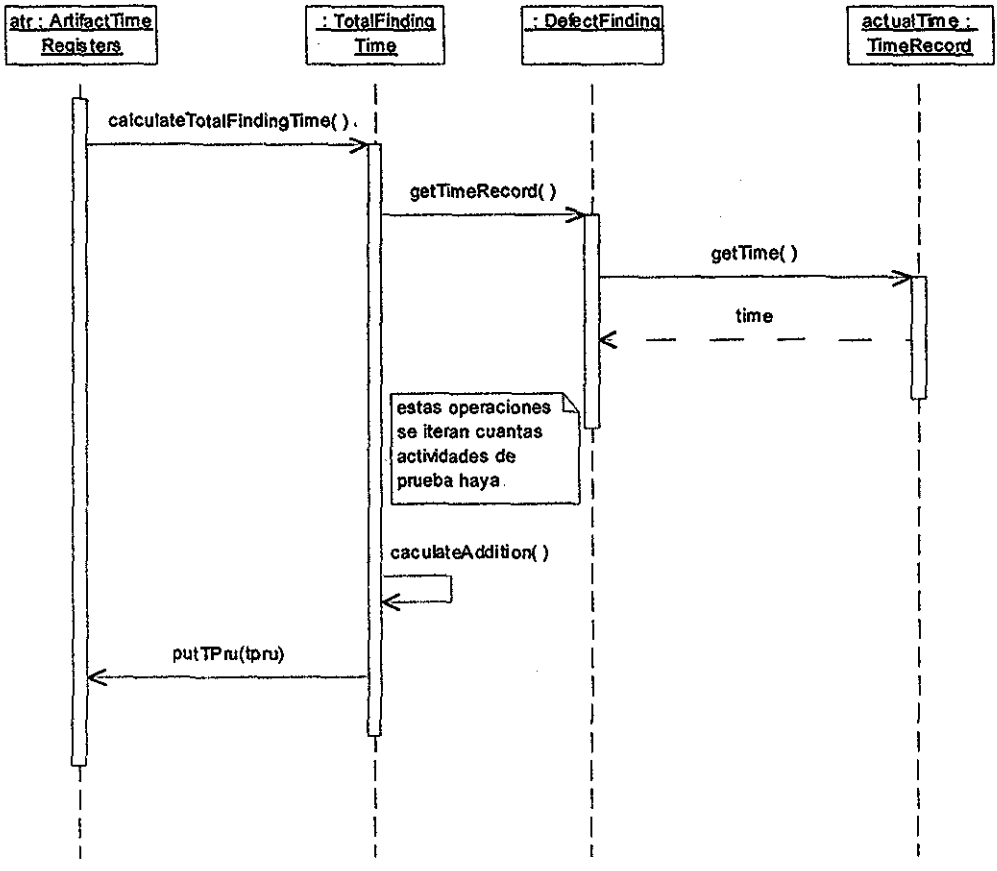


Fig. 16. Tiempo total de prueba

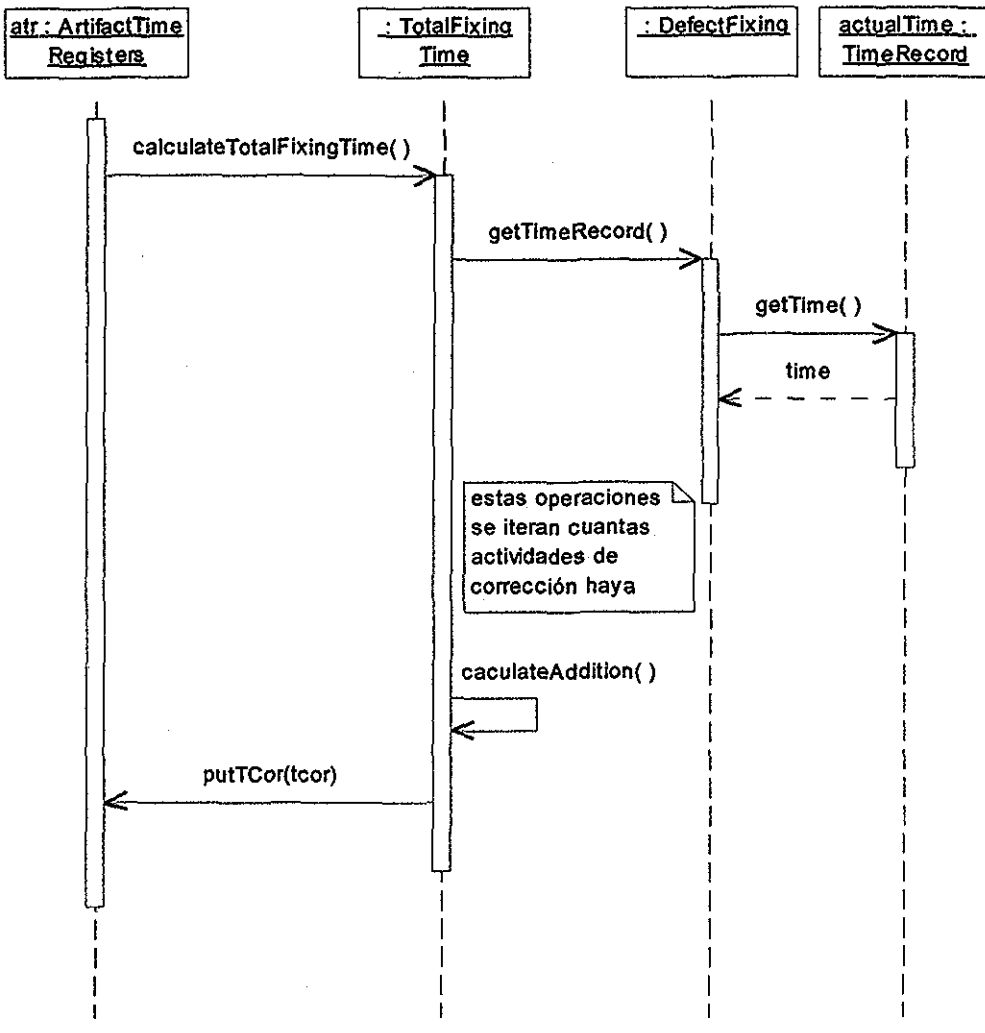


Fig. 17. Tiempo total de corrección.

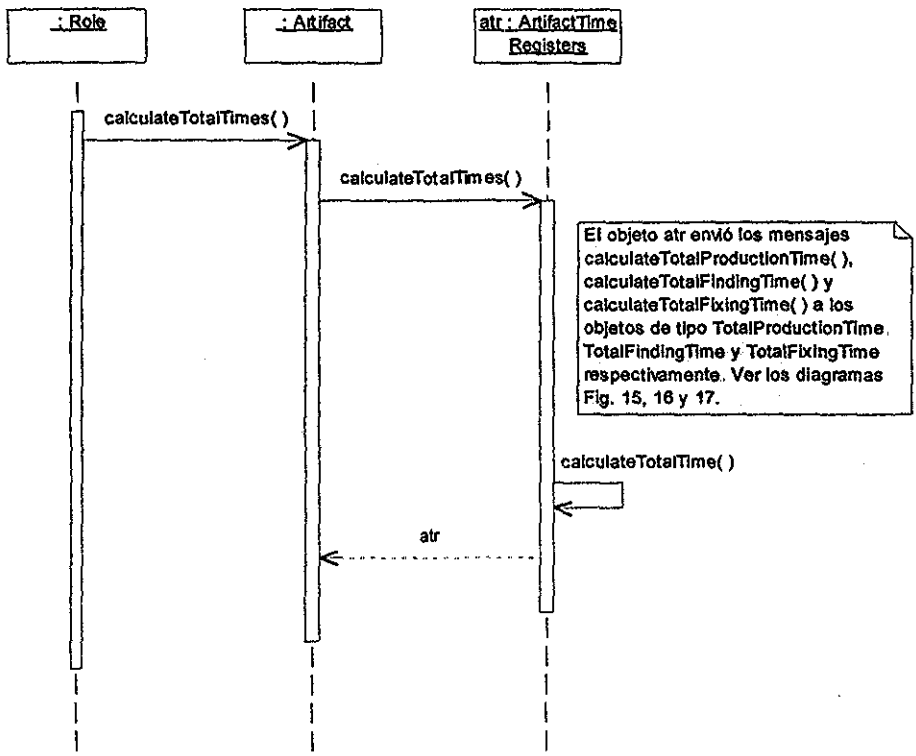


***Fig. 18.** Tiempo total del producto.*

Contexto Resultante

- Se calcula por separado el tiempo total que se invierte en las actividades de producción, pruebas y correcciones.
- Con los tiempos totales de producción, pruebas y correcciones y en función del tamaño del producto se puede estimar el tiempo que se invertirá en un proyecto futuro similar.
- La información sobre los tiempos totales de producción, pruebas y correcciones, y tamaño del producto sirve para obtener la productividad del equipo de trabajo.
- La información sobre tiempos totales de las actividades del proceso de software y el tamaño del producto ayuda a guiar hacia el mejoramiento del proceso.

CAPÍTULO 7 PATRÓN DE INDICADORES SOBRE DEFECTOS ENCONTRADOS Y CORREGIDOS.

Resumen

Este patrón calcula el número total de: defectos encontrados, corregidos, encontrados y no corregidos, encontrados y corregidos, por tamaño, encontrados por hora de pruebas, y corregidos por hora de correcciones. Utiliza los patrones **Registro de Tamaño del Producto, Registro de Búsqueda de Defectos** y **Registro de Corrección de Defectos,** tomando el registro de defectos y el tamaño del producto para hacer los cálculos necesarios.

Problema

Es necesario conocer algunos indicadores cuantitativos de los defectos encontrados y corregidos de un producto de software, como los defectos encontrados, los corregidos, los no corregidos, los defectos por tamaño, los defectos encontrados y corregidos, los encontrados por hora en pruebas y los corregidos por hora en correcciones. Estos datos permiten estimar el número de defectos que se introducirán y eliminarán en cada fase de un proyecto de software y ayudan a determinar la calidad del producto y del proceso.

Contexto

Se lleva a cabo la búsqueda y corrección de defectos sin que se registre el número de defectos encontrados y corregidos. Eventualmente existe el registro sobre información detallada de cada defecto que se encuentra y se corrige pero no se lleva un conteo sobre éstos.

Fuerzas

- El autor del producto y otros colegas hacen búsquedas y correcciones a defectos sin realizar registro alguno. Por tal motivo, es difícil conocer el número total de defectos y otros indicadores sobre defectos encontrados y corregidos.

- No se tiene información de la calidad del producto y del control de calidad de los procesos. Si se pudiera conocer esta información se podrían proponer mejoras en los procesos utilizados.

Solución

Descripción General

Finalizan las actividades de pruebas y correcciones sobre el producto, se recorre cada uno de los registros de defectos y se cuentan, así se conoce el número total de defectos encontrados (**foundDefectsCount**).

También, de estos registros se cuentan los que tienen **status** "fixed" (corregido), así se conoce el número total de defectos corregidos (**fixedDefectsCount**).

Una vez que se conocen los defectos encontrados y los corregidos se calculan los siguientes indicadores que proporcionan más información sobre ellos:

1. Defectos encontrados y no corregidos.
   nonFixedDefectsCount = foundDefectsCount - fixedDefectsCount

2. Defectos por tamaño.
   defectsForSize = foundDefectsCount / size

3. Defectos encontrados por hora.
   foundDefectsForHour = foundDefectsCount / t_{total} * 60

   Donde t_{total} es el tiempo total invertido en la elaboración de un producto y se obtuvo en el patrón **Registro de Tiempos de Producción, Pruebas y Correcciones**.

4. Defectos corregidos por hora.
   fixedDefectsForHour = fixedDefectsCount / t_{total} * 60

5. Defectos encontrados por hora de pruebas.
   foundDefectsForFindHour = foundDefectsCount / t_{pru} * 60

   Donde t_{pru} es el tiempo total invertido en la actividad de pruebas y se obtuvo en el patrón **Registro de Tiempos de Producción, Pruebas y Correcciones**.

6. Defectos corregidos por hora de correcciones.
   fixedDefectsForFixHour = fixedDefectsCount / t_{cor} * 60

Donde t_{cor} es el tiempo total invertido en la actividad de correcciones y se obtuvo en el patrón **Registro de Tiempos de Producción, Pruebas y Correcciones**.

Obtenidos todos estos indicadores, y en el momento en que el producto quede aprobado para su ingreso bajo el control de configuración, estos valores se asocian con el producto y de esta manera queda registrada información sobre los defectos encontrados y corregidos lista para su análisis posterior.

Aspectos Sociales

Los indicadores sobre defectos reflejan características personales del desarrollo del producto. Sin embargo, no deben utilizarse para evaluar el trabajo individual de la(s) persona(s) involucrada(s) en su creación.

Orientado a Objetos

Para explicar cómo calcular los indicadores se crea una clase llamada **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos), la cual se agrega a la clase **Artifact** (Artefacto). La relación estática de estas clases y de las otras que intervienen en este patrón, se presenta el diagrama de clases de la ***fig. 19***.

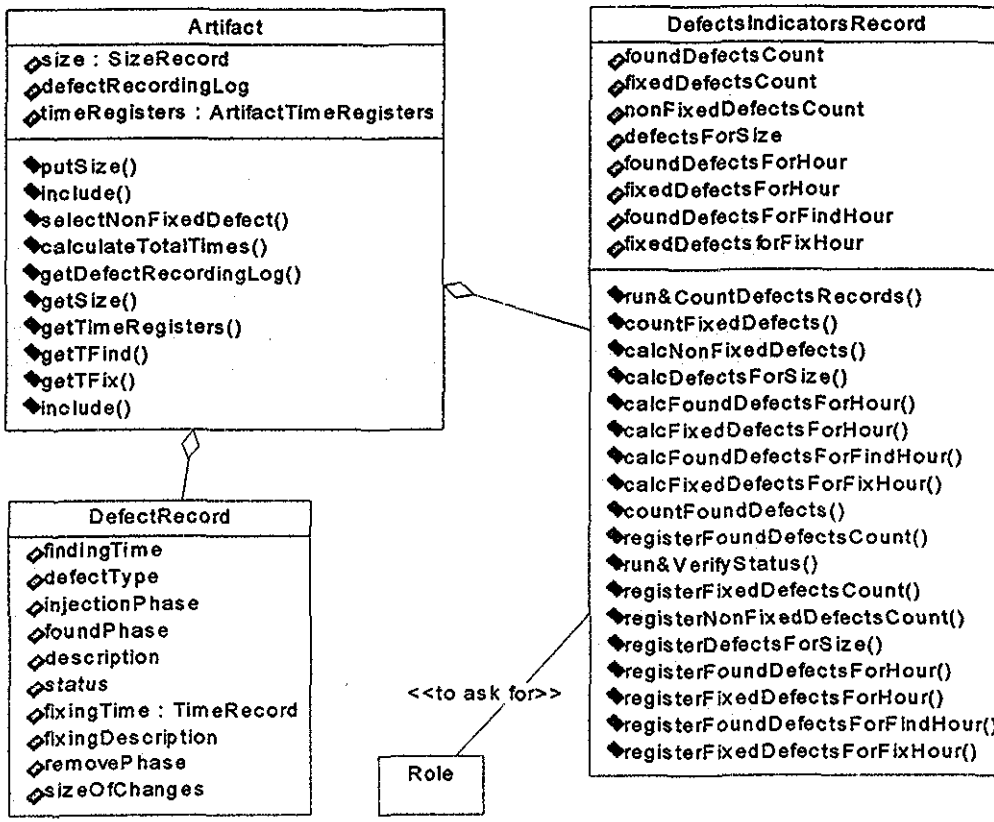


Fig. 19. Diagrama de clases del Patrón de Indicadores sobre Defectos Encontrados y Corregidos.

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) su registro de defectos representado por el atributo **defectRecordingLog** (listadoRegistroDefectos). Recorre y cuenta todos los registros individuales de defectos que haya. Cuando termina, registra el resultado del conteo en el atributo **foundDefectsCount** (contadorDefectosEncontrados), vea la ***fig. 20***.

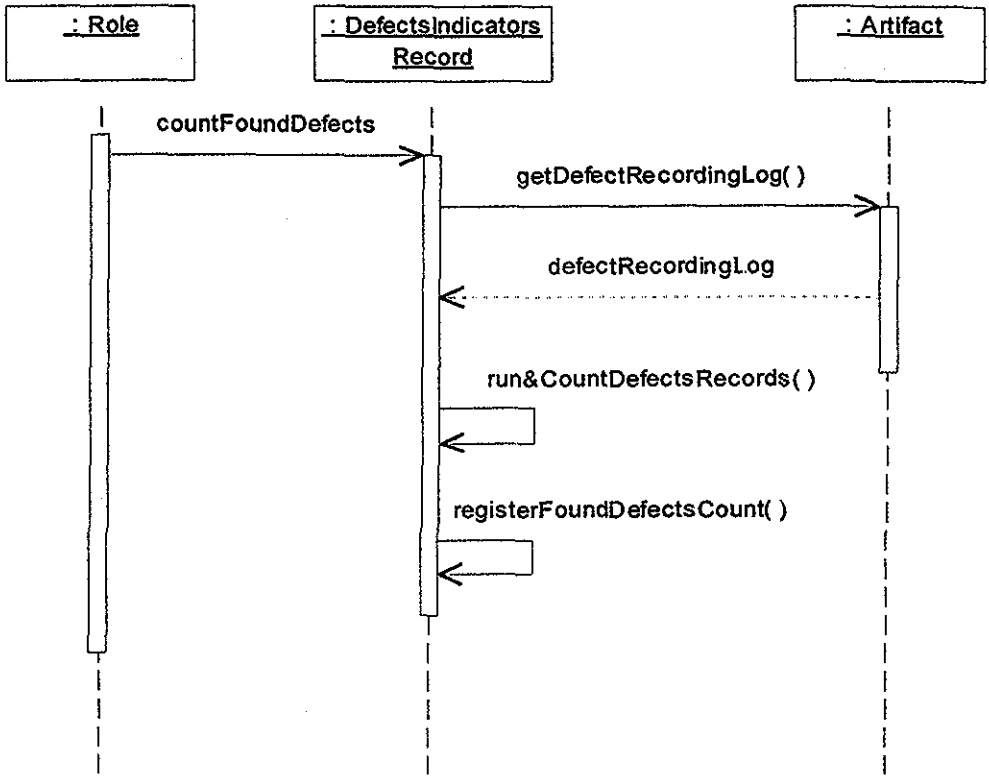


Fig. 20. *Diagrama de interacción para el conteo de defectos encontrados*

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) su registro de defectos representado por el atributo **defectRecordingLog** (listadoRegistroDefectos). Lo recorre verificando el **status** (estado) de cada uno de los registros individuales de defectos que haya y cuenta aquellos que tengan el valor **"fixed"** (corregido). Cuando termina, registra el resultado del conteo en el atributo **fixedDefectsCount** (contadorDefectosCorregidos), vea el diagrama de interacción de la ***fig. 21***.

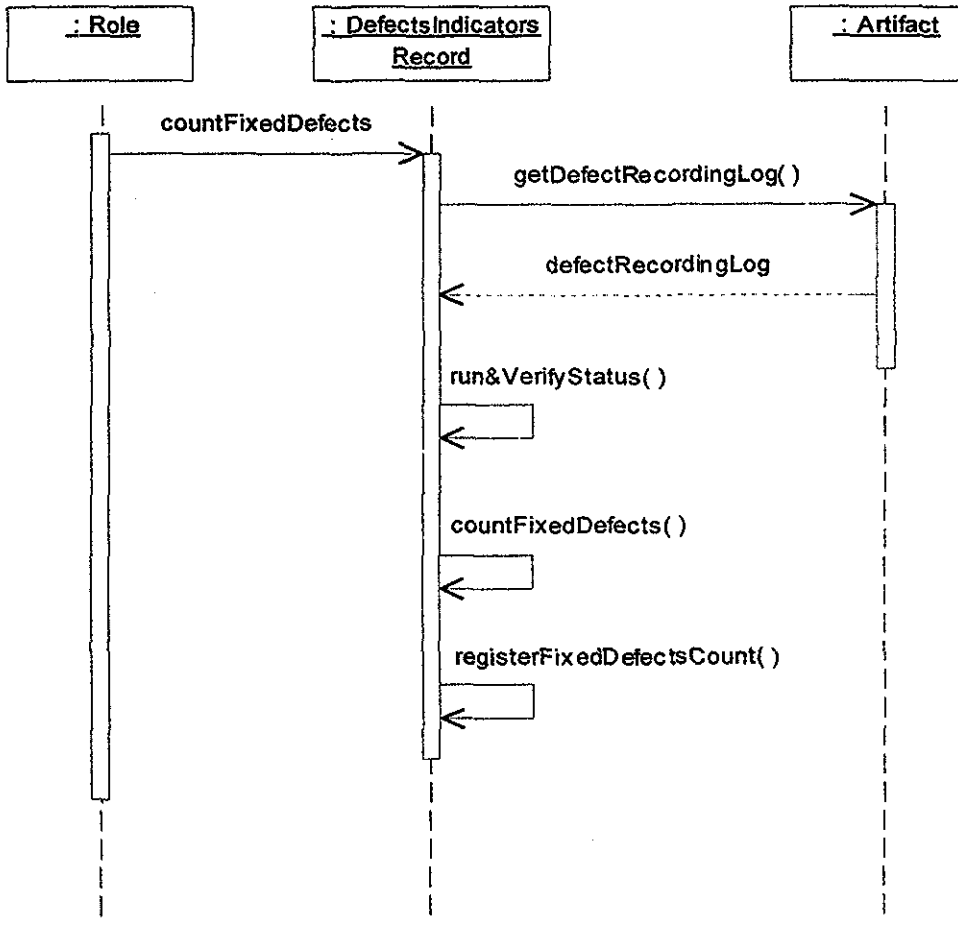


Fig. 21. *Diagrama de interacción para el conteo de defectos corregidos.*

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) tiene los atributos **foundDefectsCount** (contadorDefectosEncontrados) y **fixedDefectsCount** (contadorDefectosCorregidos). Se utiliza para realizar el cálculo de los defectos encontrados pero no corregidos, y el resultado lo registra en el atributo **nonFixedDefectsCount** (contadorDefectosNoCorregidos). Vea el diagrama de interacción de la *fig. 22*.

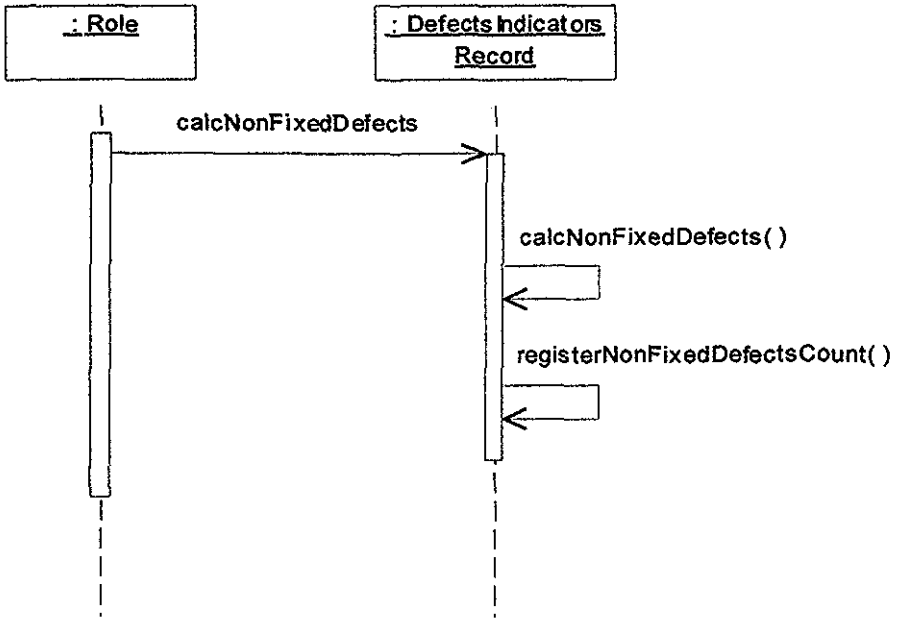


Fig. 22. Diagrama de interacción para el conteo de defectos encontrados y no corregidos.

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) el tamaño del producto, representado por el atributo **size** (tamaño). Realiza el cálculo de los defectos encontrados por unidad de tamaño utilizando también el atributo **foundDefectsCount** (contadorDefectosEncontrados). El resultado lo registra en el atributo **defectsForSize** (defectosPorTamaño). Vea el diagrama de interacción de la ***fig. 23***.

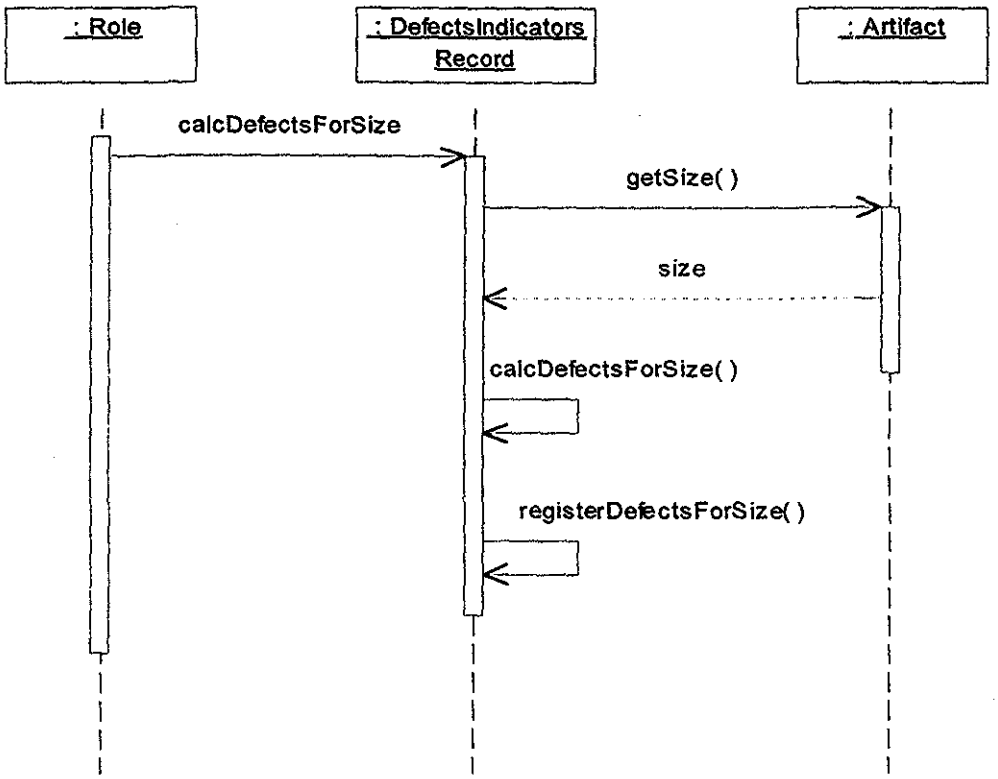


Fig. 23. Diagrama de interacción para el cálculo de defectos encontrados por tamaño.

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) el registro de tiempo total (t_{total}) del producto, representado por el atributo **timeRegisters** (registrosTiempo). Realiza el cálculo de los defectos encontrados por hora del producto utilizando también el atributo **foundDefectsCount** (contadorDefectosEncontrados). Y registra el resultado en el atributo **foundDefectsForHour** (defectosEncontradosPorHora). Vea el diagrama de interacción de la *fig. 24*.

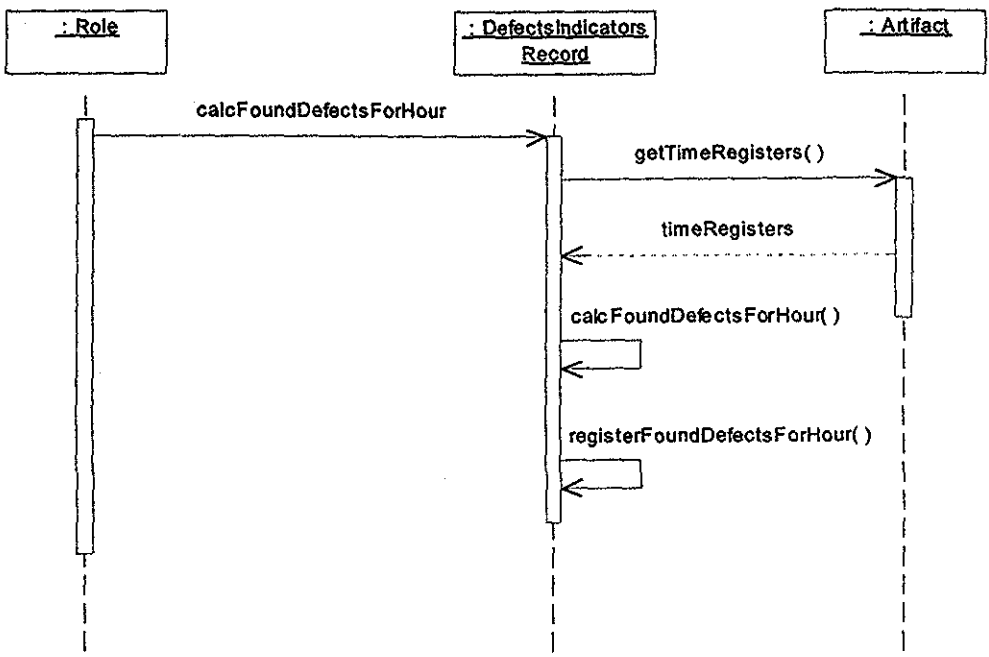


Fig. 24. Diagrama de interacción para el cálculo de defectos encontrados por hora según el tiempo total del producto.

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) el registro de tiempo total (t_{total}) del producto, representado por el atributo **timeRegisters** (registrosTiempo). Realiza el cálculo de los defectos corregidos por hora del producto utilizando también el atributo **fixedDefectsCount** (contadorDefectosCorregidos). Y registra el resultado en el atributo **fixedDefectsForHour** (defectosCorregidosPorHora). Vea el diagrama de interacción de la *fig. 25*.

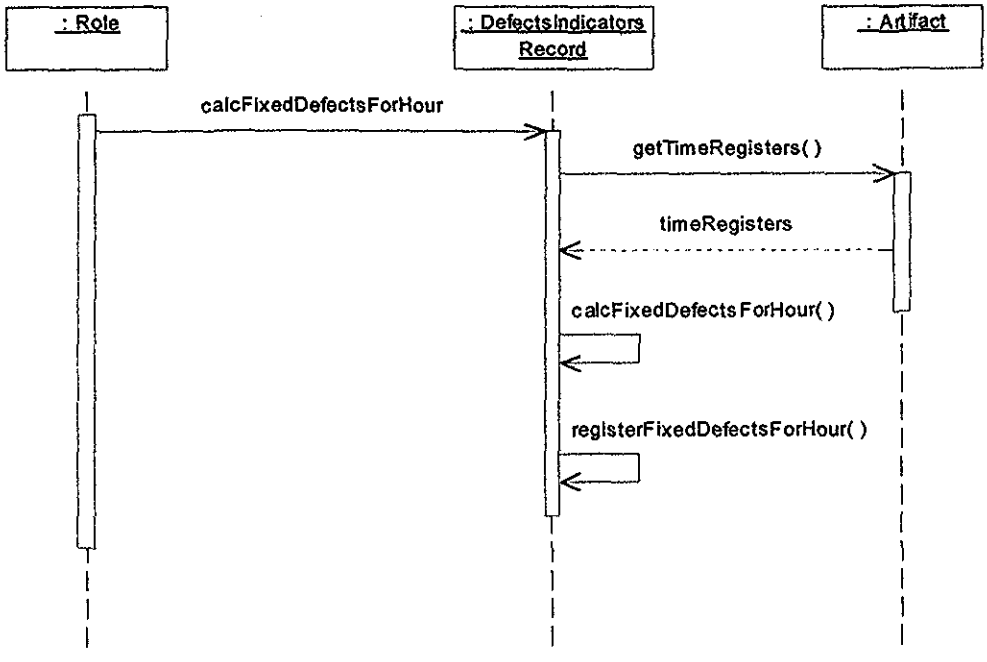


***Fig. 25.** Diagrama de interacción para el cálculo de defectos corregidos por hora según el tiempo total del producto.*

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) el registro de tiempo de pruebas del producto, representado por el atributo **tFind** (tBúsqueda). Realiza el cálculo de los defectos encontrados por hora de pruebas utilizando también el atributo **foundDefectsCount** (contadorDefectosEncontrados). Y registra el resultado en el atributo **foundDefectsForFindHour** (defectosEncontradosPorHoraBúsqueda). Vea el diagrama de interacción de la *fig. 26*.

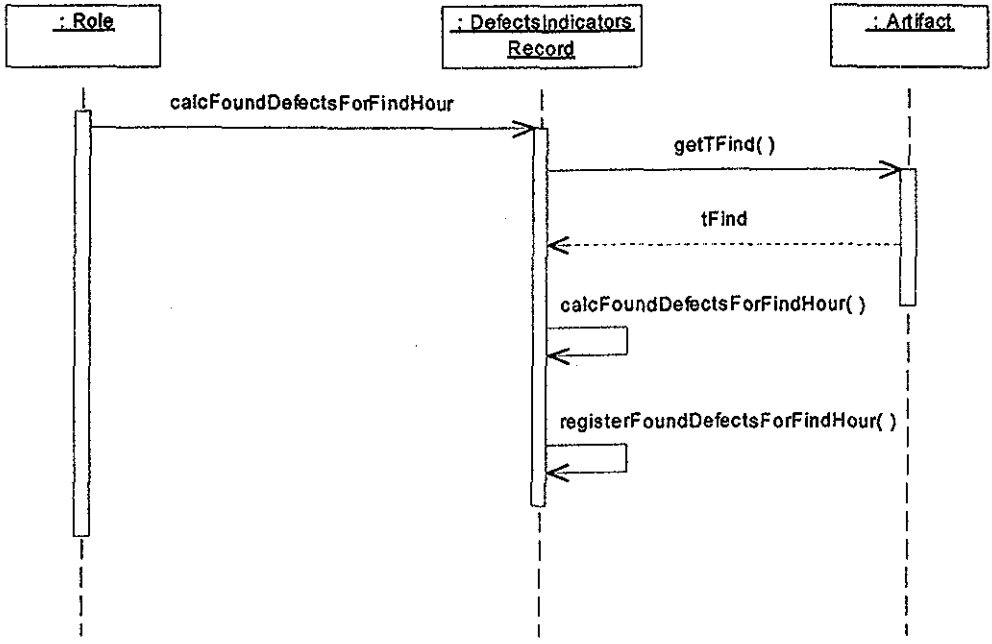


Fig. 26. Diagrama de interacción para el cálculo de defectos encontrados por hora según el tiempo total de pruebas.

La clase **DefectsIndicatorsRecord** (RegistroIndicadoresDefectos) pide a la clase **Artifact** (Artefacto) el registro de tiempo de correcciones del producto, representado por el atributo **tFix** (tCorrección), Realiza el cálculo de los defectos corregidos por hora de pruebas utilizando también el atributo **fixedDefectsCount** (contadorDefectosCorregidos). Y registra el resultado en el atributo **fixedDefectsForFixHour** (defectosCorregidosPorHoraCorrección). Vea el diagrama de interacción de la ***fig. 27***.

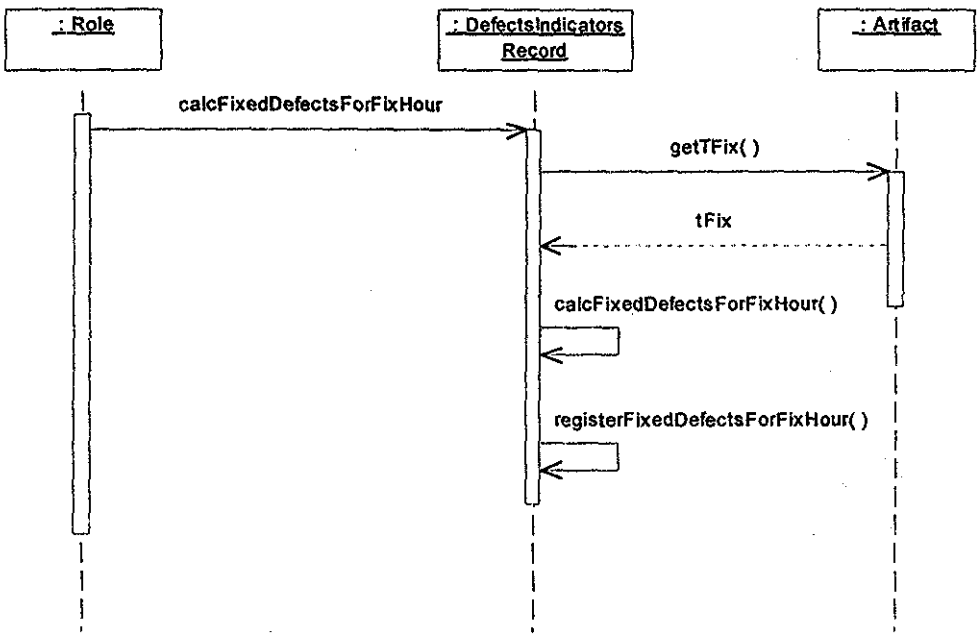


Fig. 27. Diagrama de interacción para el cálculo de defectos corregidos por hora según el tiempo total de correcciones.

Se obtienen todos los indicadores y en el momento en que el producto queda aprobado para su ingreso bajo el control de configuración, estos valores se asocian con el producto representado por la clase **Artifact**.

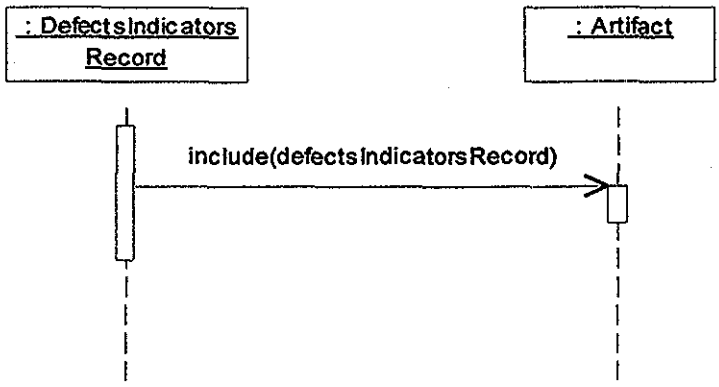


Fig. 28. Diagrama de interacción en el cual se agregan los indicadores al producto.

Contexto Resultante

- Se cuenta con información documentada de los defectos encontrados y corregidos.
- Se calcula el número total de defectos encontrados y corregidos así como otros indicadores sobre los mismos.
- Se determina la calidad del producto y de los procesos del proyecto actual.
- Se hacen estimaciones más exactas sobre el número de defectos que se introducirán y eliminarán en un proyecto futuro.
- La información sobre defectos ayuda a guiar hacia el mejoramiento del proceso.

Patrones Relacionados

Los patrones involucrados en la descripción de este patrón son el **Patrón de Registro de Tiempos de Producción, Pruebas y Correcciones** y el patrón **Product Size Record** (Registro del Tamaño de un Producto) ya que le proporcionan los registros de tiempos y de tamaño del producto. También este patrón es complementario a los patrones **Defect Finding Record** (Registro de Búsqueda de Defectos) y **Defect Fixing Record** (Registro de Corrección de Defectos) ya que agrega al registro de defectos la información relacionada con los totales de defectos encontrados y corregidos.

CONCLUSIONES

La **Guía de Métricas Básicas de Software** proporciona una forma clara y concisa de obtener información sobre tiempo invertido en las actividades del proceso de software y sobre número total de defectos. Utilizar patrones para la presentación de esta Guía resultó adecuado, por la facilidad para comunicar la existencia de un problema en un contexto dado y su solución, siendo esta última descrita en lenguaje natural y en un modelado orientado a objetos, específicamente UML.

El proceso presentado en esta **Guía** para obtener la información sobre tiempo y defectos no es un invento ni surge de la nada, está definido en el PSP (Personal Software Process) [8], el cual ya ha sido aplicado y probado sobre grupos de estudiantes que como resultado mejoraron su desempeño en la realización de programas de software.

La información que resulte de la aplicación de esta **Guía** por primera vez, no tendrá gran significado por sí misma porque no existe un punto de comparación, pero tomará mucha importancia cuando en sucesivos proyectos de software se comparen los datos de uno y de otro y se observe en dónde se encuentran las fallas, lo que permitirá proponer mejoras para aumentar la calidad de los productos de software.

Los patrones que se proponen en esta **Guía** contienen un modelado orientado a objetos por tal motivo pueden constituir un diseño preliminar para la elaboración de una herramienta de software que realice recolección y cálculo de métricas básicas para los proyectos de software. Esta automatización facilitaría a los ingenieros de software la tarea de registrar con exactitud y sin ningún olvido el tiempo que se invierte en cada una de las actividades tanto de producción como de pruebas y de correcciones.

Esta **Guía** está dirigida hacia las organizaciones que desarrollan sistemas de software para que la utilicen como auxiliar en la obtención de información cuantitativa de tiempo y de defectos de las actividades del proceso de software; proporcionándoles de esta manera, beneficios en la planeación de sus proyectos, mejoras en la calidad de sus productos así como en las técnicas utilizadas para producirlos, probarlos y corregirlos, y estimaciones más certeras.

El análisis y la evaluación de las métricas básicas que se obtienen en los patrones descritos en esta **Guía** proporcionan beneficios encaminados hacia el mejoramiento del proceso de software. Por ejemplo, en el primer patrón, **Registro de Tiempos de Producción, Pruebas y Correcciones**, se obtienen las siguientes métricas básicas:

- Tiempo total de producción.

- Tiempo total de prueba.
- Tiempo total de corrección.
- Tiempo total del producto (Que es la suma de las anteriores).

Y en el patrón **Registro del Tamaño del Producto [10]** se obtiene la métrica básica **Tamaño Total del Producto**. El análisis de esta información permite hacer planes más apegados a lo real para proyectos futuros similares, lo que significa terminar el trabajo en el tiempo establecido, es decir, se toman los datos de tiempo y en función del tamaño total del producto se estima el tiempo total que se invertirá en la realización de ese proyecto. Estos planes proporcionan una línea base sobre la cual se mide la elaboración de un producto de software mostrando el tiempo que se invierte en cada fase del proceso de software. Si los planes contienen estimaciones más exactas entonces los costos esperados para hacer el trabajo corresponderán con los reales.

Por otro lado, en el segundo patrón, **Indicadores sobre Defectos Encontrados y Corregidos**, se obtienen las siguientes métricas básicas:

- Número de defectos encontrados.
- Número de defectos corregidos.
- Número de defectos encontrados y no corregidos.
- Número de defectos encontrados por tamaño.
- Número de defectos encontrados por hora según el tiempo total del producto.
- Número de defectos corregidos por hora según el tiempo total del producto.
- Número de defectos encontrados por hora según el tiempo total de pruebas.
- Número de defectos corregidos por hora según el tiempo total de correcciones.

El análisis de esta información permite hacer estimaciones más exactas sobre el número de defectos que se introducirán y eliminarán en cada fase del proyecto de programación. Si se hace una comparación de los datos del proyecto actual con la experiencia histórica se puede determinar la calidad del programa que se va a desarrollar y permite estimar las pruebas y revisiones de código que se llevarán a cabo para ver que los programas estén libres de defectos.

Los patrones básicos **Registro de Búsqueda de Defectos** [10] y **Registro de Corrección de Defectos** [10], proporcionan una lista con los detalles sobre los defectos encontrados, por ejemplo, tiempo que se invirtió para encontrarlo, tipo de defecto, fase en la que fue introducido, fase en la que fue encontrado, su descripción, un estado indicando si ya fue corregido o no, tiempo que se invirtió para corregirlo, descripción de la corrección, fase en la que se corrigió, entre otros.

Esta información permite analizar y determinar la manera de prevenir o encontrar los defectos que se introducen evitando que se vuelvan a cometer y facilitando su localización al momento de buscarlos para su corrección.

La obtención de estas métricas básicas y su análisis también proporcionan los siguientes beneficios:

- **Mejorar la programación**. Los datos de defectos ayudan a entender sus causas y cómo se pueden proponer mejoras a la forma de escribir programas.

- **Reducir el número de defectos en los programas**. Todo ingeniero de software está expuesto a introducir defectos en los programas en cualquier momento, sin embargo, contando con información sobre defectos, como los indicadores que se obtienen en este patrón, se puede definir un método adecuado para reducir el número de defectos que se introducen.

- **Ahorrar tiempo**. Es importante eliminar los defectos tan pronto como sea posible, es decir, inmediatamente después de que se introducen; si desde los requerimientos hay errores entonces es seguro que tanto el diseño como la implementación también los tendrán, en consecuencia se invierte más tiempo en encontrar y corregir los defectos introducidos.

- **Ahorrar dinero**. Los defectos resultan costosos después de la prueba de unidad. Los costos de buscarlos y corregirlos se incrementan 10 veces con cada prueba subsecuente o fase de mantenimiento.

- **Trabajar responsablemente**. Los defectos son introducidos por los ingenieros y es su responsabilidad buscarlos y corregirlos [8].

Las métricas básicas de defectos también son útiles para estimar el número de defectos esperados en un nuevo programa a realizar; PSP (Personal Software Process) [8] describe este dato como defectos por mil líneas de código (KLOC) y lo llama densidad de defectos (Dd), la cual se mide en unidades de defectos/KLOC, donde la K significa 1000. Para calcular los defectos/KLOC totales encontrados en un programa se hace lo siguiente [8]:

1. Sumar los números totales de defectos (D) encontrados en todas las fases del proceso.

2. Contar el número (N) de LOC en el programa.

3. Calcular los defectos por KLOC como Dd = 1000*D/N.

Por ejemplo, si un programa de 96 LOC tuvo un total de 14 defectos entonces la densidad de defectos sería 1000*14/96 = 145.83 defectos/KLOC.

El cálculo de la densidad de defectos es útil e importante siempre y cuando se utilicen las métricas correctas de tamaño del producto.

La planeación de un programa requiere primero de estimar el número de líneas de código que tendrá y luego calcular el porcentaje de defectos/KLOC de los programas previamente desarrollados. Con estos números se puede calcular el número esperado de defectos/KLOC en el nuevo programa de la siguiente manera [8]:

$$Dd_{plan} = 1000(D_1+.....+D_i)/(N_1+.....+N_i)$$

Donde:

i = Número de programas anteriores.
D = Número de defectos.
N = Número de líneas de código.

Supóngase, por ejemplo, que se tienen los datos de 5 programas que se muestran en la siguiente tabla:

Número de Programa	Defectos(D)	LOC(N)
1	6	37
2	11	62
3	7	49
4	9	53
5	5	28
Total	**38**	**229**

Ahora, el valor de Dd_{plan} se calcula como sigue [8]:

Dd_{plan} = 1000*(6+11+7+9+5)/(37+62+49+53+28)
= 1000*38/229 = 165.94 defectos/KLOC

Si se asume que este nuevo programa tendrá esta densidad de defectos entonces se calcula su número esperado de defectos como [8]:

$$D_{plan} = N_{plan}*Dd_{plan}/1000$$

Donde:

N_{plan} = Número de líneas de código estimadas.
Dd_{plan} = Densidad de defectos.

Ahora, utilizando este mismo ejemplo y asumiendo que las líneas de código estimadas del nuevo programa sean 56, se calcula el número esperado de defectos como [8]:

D_{plan} = 56+165.94/1000
= 9.29 defectos

Estos datos nos indican que para un programa con 56 líneas de código planeadas se espera tener 9 defectos.

Como se puede observar, el objetivo principal de las métricas de software es proporcionar un elemento de análisis para crear productos de calidad, lo cual se logra con la experiencia histórica, y PSP (Personal Software Process) [8] es una guía que indica cómo reunir datos de tiempo y de defectos y cómo usarlos para mejorar el proceso y la calidad de los productos. Entre más información se tenga de los proyectos anteriores se obtienen proyectos actuales más exitosos.

REFERENCIAS

1. Booch Grady, Rumbaugh James, Jacobson Ivar; *The Unified Modeling Language User Guide*; Editorial Addison-Wesley; 1999.

2. Buschmann Frank, Meunier Regine, Rohnert Hans, Sommerlad Peter, Stal Michael; *A System of Patterns*; Editorial Jonn Wiley & Sons; 1996.

3. Coplien James, Tutorial: Foundation of Pattern Concepts and Pattern Writing; XV Brazilian Symposium on Software Engineering/SBES; Octubre 2001.

4. DeLano, D. and Rising, L. Patterns for System Testing, A Pattern Language for Pattern Writing. *Patterns Languages of Program Design 3*. Martin, R., Riehle D. and Buschmann, F. Eds. Software Patterns Series Addison-Wesley, 198, 503-525.

5. Fenton Norman E., Lawrence Pfleeger Shari; *Software Metrics*, A Rigorous and Practical Approach; Segunda Edición; Editorial PWS Publishing Company; 1997.

6. Gamma, B., R. Helm, R. Johnson, and J. Vlissides; *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley; 1995.

7. Humphrey Watts S.; *A Discipline for Software Engineering*; Editorial Addison-Wesley;1995.

8. Humphrey Watts S.; *Introduction to the Personal Software Process*; Editorial Addison-Wesley; 1997.

9. Humphrey Watts S.; *Introduction to the Team Software Process*; Editorial Addison-Wesley; 2000.

10. Oktaba Hanna, Ibargüengoitia González Guadalupe; *Basic Measures Patterns for Software Process Improvement*; Nith International Conference on Software Quality Proceedings; October 4-6; 1999; Cambridge, Massachusetts.

11. Pressman Roger S.; *Ingeniería de Software*, un Enfoque Práctico; Cuarta Edición; Editorial McGraw Hill; 1998.