

Instituto Tecnológico y de Estudios

Superiores de Monterrey

Campus Guadalajara

**Escuela de Graduados en Ingeniería y Arquitectura (EGIA)**

**Maestría en Ciencias de la Computación**

**TÉCNICAS DE DETECCIÓN DE DEFECTOS: BUG MANAGER**

**AUTOR**: **Eduardo Campos Peiro**

**ASESORES**: Dr. Oscar Adrián Mondragón Campos

Guadalajara (Jal), 4 de Septiembre de 2011

Agradecimientos

Quiero comenzar por agradecer al Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Guadalajara, por la facilidad y oportunidad de realizar la Maestría en Ciencias de la Computación. Quiero agradecer también a mis compañeros de dicho posgrado. Gracias por los buenos momentos vividos durante estos dos años y medio que estuvimos compartiendo conocimientos, experiencias, pero sobre todo amistad, compañerismo y valores. Me llevo grandes experiencias, anécdotas y maneras de ver la vida de cada uno de ustedes. De verdad me considero afortunado por haber conocido y compartido con este gran grupo de personas cada una de las materias recibidas durante el posgrado.

Un agradecimiento de manera especial a mis compañeros de tesis: Marco Antonio Rangel Bocardo y Humberto García Robles. Sinceramente creo que se formó un excelente grupo de trabajo. Pudimos colaborar verdaderamente como equipo, haciendo que las reuniones, discusiones y reparticiones de actividades fueran eficientes y eficaces. También agradecer a Humberto y su familia el haber facilitado su domicilio para realizar las reuniones semanales. Sé que no es fácil recibir personas constantemente y por eso les estoy agradecido sinceramente.

También un agradecimiento especial a mí asesor de tesis, el Dr. Oscar Adrián Mondragón Campos, ya que simple y sencillamente fungió como tal de principio a fin. Gran asesor que con su experiencia y conocimiento contribuyó de manera significativa en todo el proceso de desarrollo de este trabajo de investigación. Cada vez que algún obstáculo o dificultad surgía, él me apoyaba y guiaba para encontrar la mejor solución posible. Por supuesto, una mención especial merece el Instituto Jalisciense de la Juventud (IJJ), que al haberme otorgado un apoyo financiero hizo posible la realización del posgrado y de este trabajo de tesis. Sin ellos simplemente no hubiera sido posible.

Finalmente, el agradecimiento más especial a mis padres Carlos Eduardo Campos y Libertad Peiro y a mi compañera de vida María Fernanda García, que día con día me apoyaron de manera incondicional y de corazón, dándome ánimos para alcanzar esta meta. Gracias a ustedes tres por el amor, el apoyo, los principios y la educación que me dieron y que hicieron posible que el día de hoy me encuentre escribiendo estas palabras.

Resumen

La administración de la calidad en el desarrollo de sistemas de software ha sido un tema de debate y discusión desde hace ya varios años. Esto ha provocado que se definan un gran número de estándares y modelos, así como que se desarrollen decenas de herramientas de software para mejorar la calidad de los productos finales. La adopción de modelos, procesos, herramientas y estándares no ha sido fácil ni muy aceptada por un gran número de empresas dedicadas a este rubro. Las razones principales son:

* La creencia que las actividades de calidad solamente provocan un retraso en la entrega de los proyectos, al ser actividades que consumen un tiempo considerable.
* Que los modelos y procesos son complicados y muy elaborados para implementarse, especialmente por empresas que no cuentan con grandes recursos humanos ni económicos.
* Que las herramientas de software solamente cuentan con una funcionalidad parcial, y por lo tanto solo atacan una parte del problema.

El trabajo de investigación aborda estos problemas mediante el Bug-Manager (BM). El BM permite la creación de un plan de calidad, el registro de técnicas de detección de defectos mediante plantillas estándares y personalizadas, el registro, caracterización y seguimiento de defectos, y la generación de gráficas y estadísticas con información importante acerca del proceso de desarrollo.

La investigación se centra en la descripción de las diferentes técnicas de detección de defectos, sus ventajas y desventajas, los motivadores y desmotivadores que favorecen o condicionan su adopción e implementación, e inclusive las condiciones y características que generan una implementación exitosa de estas técnicas. Finalmente, se explica cómo es que estas técnicas contribuyen a mejorar la calidad y porqué son importantes para el aseguramiento de la misma.

El desarrollo del BM no pretende ser la panacea ni sustituir todas las herramientas existentes, sino ser una herramienta que facilite la implementación de actividades de calidad, especialmente en empresas sin grandes recursos, proveyendo funcionalidades para la administración eficiente de los elementos más importantes de la calidad de software.

Contenido

Agradecimientos i

Resumen iii

Lista de Tablas viii

Lista de Figuras ix

1. Introducción 1

1.1 Planteamiento del problema 1

1.2 Objetivos y alcances 2

1.3 Contribuciones de la investigación 4

1.4 Organización del documento 4

2. Calidad de Software 6

2.1 ¿Qué es calidad? 6

2.2 Calidad aceptable 8

2.3 La importancia de la calidad 10

2.3.1 Costos de la calidad de software 10

2.3.2 Revisiones 12

3. Implementación de la Calidad 15

3.1 Modelos de calidad 16

3.1.1 ISO 9126 16

3.1.2 CMMI-DEV 18

3.1.3 Boehm et al 22

3.1.4 Clasificación, situación actual y escenarios de los modelos de calidad 25

3.2 Herramientas de software 29

3.2.1 Bugzilla 32

3.2.2 GanttProject 33

3.2.3 Rational Clear Quest 34

4. Conociendo la Calidad 37

4.1 Métricas de producto 37

4.2 Métricas de proceso 39

4.2.1 Rendimiento 40

4.2.2 Costo de la calidad 41

4.2.3 Razón de revisión 42

4.2.4 Razón de fase 43

4.2.5 Índice de calidad de proceso 43

5. Técnicas de Detección de Defectos 46

5.1 Tipos de técnicas 49

5.1.1 Revisión Personal 49

5.1.2 Revisión entre colegas 50

5.1.3 Caminata 51

5.1.4 Inspección 51

5.2 Implementación de técnicas de detección de defectos 53

5.3 Resultados de las técnicas de detección de defectos 57

6. Bug-Manager 62

6.1 Metodología 62

6.2 Arquitectura 64

6.3 Características Técnicas 64

6.4 Módulos 65

6.5 Funcionalidades y objetivos 65

6.6 Supuestos y restricciones 67

6.7 Pantallas 68

7. Resultados Obtenidos 69

8. Conclusiones 70

Referencias 71

A. Apéndice A 75

B. Apéndice B 76

Vitae 77

Lista de Tablas

[Tabla 2.1 Comparación de costos de calidad. 13](#_Toc315699373)

[**Tabla 3.1. Áreas de proceso de niveles de CMMI-DEV**. 20](#_Toc315699374)

[Tabla 3.2. Sistemas de control de versiones. 30](#_Toc315699375)

[**Tabla 4.1. Densidad de defectos por nivel de CMMI**. 39](#_Toc315699376)

[Tabla 5.1. Motivadores de las técnicas de detección de defectos. 54](#_Toc315699377)

[Tabla 5.2. Desmotivadores de las técnicas de detección de defectos. 55](#_Toc315699378)

[Tabla 5.3. Número de revisores contra efectividad. 59](#_Toc315699379)

[Tabla 5.4. Concentrado resultados de revisiones. 60](#_Toc315699380)

[Tabla 5.5. Comparación entre revisiones y pruebas. 60](#_Toc315699381)

[Tabla 6.1. Características técnicas del sistema BM. 65](#_Toc315699382)

[Tabla 6.2. Funcionalidades del sistema BM. 66](#_Toc315699383)

Lista de Figuras

[Figura 2.1. Costos de la calidad. 11](#_Toc306352857)

[Figura 2.2. Modelo de calidad de software. 12](#_Toc306352858)

[Figura 2.3. Relación costo-calendario. 14](#_Toc306352859)

[Figura 3.1. Modelo ISO 9126 para calidad interna y externa. 17](#_Toc306352860)

[Figura 3.2. Modelo ISO 9126 para calidad en uso. 17](#_Toc306352861)

[Figura 3.3. Niveles de madurez de CMMI. 19](#_Toc306352862)

[Figura 3.4. Modelo teórico sobre la reducción de costos de calidad. 22](#_Toc306352863)

[Figura 3.5. Árbol de características de calidad de software. 25](#_Toc306352864)

[Figura 3.6. Clasificación de modelos de calidad de software. 26](#_Toc306352865)

[Figura 3.7. Búsqueda de defectos en Bugzilla. 33](#_Toc306352866)

[Figura 3.8. GanttProject. 34](#_Toc306352867)

[Figura 3.9. Rational Clear Quest. 36](#_Toc306352868)

[Figura 4.1. Rendimiento de proceso 41](#_Toc306352869)

[Figura 4.2. Relación A/FR y defectos por KLOC en prueba unitaria. 42](#_Toc306352870)

[Figura 4.3. Índice de proceso de calidad. 44](#_Toc306352871)

[Figura 5.1. Probabilidad de detectar defectos respecto al número de revisores. 49](#_Toc306352872)

[Figura 5.2. Tipos de revisión entre colegas. 51](#_Toc306352873)

[Figura 5.3. Efectividad de las revisiones. 58](#_Toc306352874)

[Figura 5.4. Efectividad promedio de revisiones y pruebas. 60](#_Toc306352875)

[Figura 5.5. Eficiencia promedio de revisiones y pruebas. 61](#_Toc306352876)

[Figura 6.1. Arquitectura modelo-vista-controlador. 64](#_Toc306352877)

[Figura 6.2. Módulos del sistema BM. 65](#_Toc306352878)

CAPÍTULO 1

1. Introducción
   1. Planteamiento del problema

La administración de la calidad en el desarrollo de sistemas de software es un rubro dentro de la industria que no ha logrado una gran aceptación, y que por lo tanto no se implementa como debería o se esperaría de él, en comparación con otras ramas de la ingeniería con procesos y modelos de calidad bastante desarrollados y eficientes. El hecho de que el software esté presente en prácticamente todas las actividades que realizamos hoy en día, hace que el tema de la calidad en el mismo tome gran relevancia.

De 700 evaluaciones sobre el Capability Maturity Model (CMM) realizadas alrededor del mundo entre 1992 y 1997 que fueron reportadas al Software Engineering Institute (SEI), solamente 20 empresas resultaron con un nivel de madurez 4 o 5, es decir, menos del 3% [[1](#PJa00)]. Además, solamente el 20% de las empresas realiza con regularidad alguna técnica de revisión [[2](#PJo98)]. Esto nos lleva a preguntarnos los motivos por los cuales la gran mayoría de empresas dedicadas al desarrollo de software, no establecen algún tipo de proceso, modelo o simplemente actividades que contribuyan a incrementar la calidad global de sus productos.

Dentro de los motivos principales se encuentran la falta de tiempo, el costo, la falta de recursos humanos, la complejidad en su implementación, la falta de entrenamiento adecuado al personal y el síndrome conocido como “No aplica aquí” (NAH, por sus siglas en inglés) [[3](#Har05)]. Este síndrome representa un fenómeno que llama la atención, y que básicamente consiste en que la mayoría de las empresas creen que la administración de la calidad es algo hecho para empresas como **International Business Machines (IBM) o Hewlett Packard (HP)**, pero que no aplica en el contexto de negocio de éstas. Es decir, **este gran** número de empresas cree en las estadísticas y la literatura publicada, sin embargo no creen que la implementación de actividades de calidad tenga los mismos resultados que tuvo en otras empresas [[4](#PJa98)].

Todo este tipo de factores y desmotivadores llevan a la falsa creencia que las prácticas de calidad hacen más lento y complicado el proceso de desarrollo, provocando

un retraso en la entrega del producto, además de incrementar su costo. Sin embargo, la implementación de estas prácticas tiene un efecto contrario a estas falsas creencias. Realizar alguna técnica de detección de defectos, como la inspección, si es bien aplicada, puede llegar a ser de 2 a 4 veces más efectiva que la etapa de pruebas; y en algunos casos encontrar hasta el 90% de los errores antes de correr la primera prueba [[5](#GCh99)].

Evidentemente esto tiene como consecuencia la reducción del tiempo invertido en la etapa de pruebas. Por otro lado, el costo de corregir un defecto en las etapas de diseño o codificación, en promedio, es de 1 a 2 órdenes de magnitud menor que si se corrigiera en la etapa final de pruebas o una vez que el software salió a producción [[2](#PJo98)]. Lo que muchas empresas no tienen en cuenta es que la calidad en el desarrollo debería ser la prioridad. Implementar calidad significa que los productos se vuelvan predecibles en tamaño, esfuerzo y costo de desarrollo, que sean más fáciles de dar seguimiento y que tengan un costo menor de mantenimiento, al contar con menos defectos que salen a producción.

A su vez, el tiempo de desarrollo se reduce al llegar a la etapa de pruebas con menos defectos, lo que aumenta la probabilidad de entregar el producto en tiempo y forma. Pero sobre todo, y lo más importante, es que de esta manera se está más cerca de conseguir la satisfacción del cliente, lo que en última instancia provoca el desarrollo de más y mejores proyectos.

* 1. Objetivos y alcances

En la actualidad existe un gran número de sistemas que apoyan en la implementación y seguimiento de actividades de calidad. Algunos sistemas se enfocan principalmente en el registro y seguimiento de defectos del sistema en desarrollo, en otros su principal funcionalidad radica en la versatilidad al momento de configurar cómo se registrarán los defectos. Algunos más cuentan con la capacidad de integrarse con planes de pruebas para llevar un mejor control de lo que ocurre en estas etapas [[6](#Wik11)]. Dentro de estos sistemas, se encuentran los que son propietarios y requieren forzosamente de alguna licencia y también los que se rigen bajo la “Licencia Pública General” (GPL, por sus siglas en inglés).

El común denominador de estos sistemas es que son muy eficientes en determinados procesos y etapas del ciclo de vida, pero dejan de serlo para otras fases igual de importantes en el desarrollo. Esto, aunado a los desmotivadores previamente mencionados y a la creencia general sobre la dificultad existente para implementar modelos y planes de calidad, inhibe a las empresas para adoptar actividades que mejoren la calidad final de sus productos de software.

El trabajo de investigación se centra en el desarrollo de un sistema de software que facilite la adopción e implementación de actividades de calidad en las empresas, especialmente pequeñas y medianas, que no cuenten con estas actividades mediante:

* La posibilidad de registrar las actividades de desarrollo y de calidad asociadas a las primeras, para establecer el esfuerzo requerido en la realización del proyecto.
* Registrar, caracterizar y dar seguimiento a los defectos surgidos durante el proyecto.
* La posibilidad de llevar a cabo diferentes técnicas de detección de defectos asociadas a actividades de calidad, para conocer el esfuerzo invertido en estas actividades y la eficiencia de la implementación de las técnicas.
* Llevar a cabo las técnicas de detección de defectos mediante listas de chequeo personalizadas, con el objetivo de incrementar la eficiencia de las mismas de manera paulatina.
* Generar estadísticas y métricas que permitan conocer al equipo de desarrollo, líder de proyecto y gerente cómo se desarrolló a final de cuentas el proyecto. Todo esto a partir de los datos introducidos en el registro de las actividades de desarrollo **y de** calidad, del registro de defectos y de las técnicas de detección de defectos.

También se pretende con la investigación exponer un panorama detallado del estado del arte en cuanto a técnicas de detección de defectos se refiere. Específicamente, describir cada una de las diferentes técnicas, exponer sus ventajas y desventajas, mencionar los motivadores y desmotivadores que ocasionan que se lleven a cabo o no, qué tan eficientes pueden llegar a ser, y finalmente, establecer cómo es que estas técnicas colaboran para que los sistemas de software sean **de mayor** calidad.

* 1. Contribuciones de la investigación

**La primera contribución es** la explicación detallada y la concientización sobre la importancia y el papel de las técnicas de detección de defectos en el aseguramiento de la calidad en productos de software. Cómo es que estas técnicas correctamente implementadas contribuyen a disminuir el número total de defectos y la densidad de los mismos. Además cómo estas técnicas reducen el tiempo total invertido en las fases de pruebas y mantenimiento, lo que se traduce en la reducción del costo total de elaboración del producto.

**La segunda contribución consiste en** la exposición sobre cómo el sistema BM podría colaborar y facilitar la adopción y realización de actividades de calidad en empresas dedicadas al desarrollo de software, mediante el uso del sistema, y entendiéndolo como un apoyo y no una solución única y total a los problemas específicos de cada empresa. Es decir, cómo el sistema podría atacar y reducir los desmotivadores y argumentos principales por los que las empresas no cuentan con algún plan de aseguramiento de la calidad, para así facilitar la adopción del plan.

* 1. Organización del documento

El resto del documento se organiza de la siguiente manera. En el capítulo 2 se provee el significado de la calidad en el software, qué significa calidad aceptable y porqué es importante abordar la calidad en todo el proceso de desarrollo. En el capítulo 3 se expone cómo es que se puede implementar la calidad durante este proceso, mientras el cuarto expone cómo se puede mejorar paulatinamente. El capítulo 5 se centra en la exposición de las diferentes técnicas de detección de defectos, sus ventajas y desventajas y su contribución al mejoramiento de la calidad. Este capítulo corresponde al tema de especialización de la tesis.

En el capítulo 6 se explica qué es el BM, haciendo un énfasis en las técnicas de detección de defectos, y cómo se desarrolló. El capítulo 7 muestra los resultados obtenidos de la investigación teórica y del desarrollo del sistema. El capítulo 8 expone las conclusiones surgidas del desarrollo de la investigación y del sistema, así como sugerencias sobre trabajo futuro que podría agregar más valor a lo ya hecho. El capítulo 9 lista las referencias bibliográficas utilizadas durante la investigación y el capítulo 10 contiene los anexos que complementan este documento de tesis.

CAPÍTULO 2

1. Calidad de Software
   1. ¿Qué es calidad?

La ingeniería de software es una actividad considerada (hasta el momento) como una de las más complicadas para la elaboración de productos finales con calidad. Sin duda alguna, realizar productos con calidad es cada vez más importante, debido a la trascendencia que adquiere día con día el software en las actividades y procesos diarios de empresas. Es por eso que no sólo es importante conocer el concepto de calidad en términos generales, sino conocer las particularidades de la ingeniería de software, para adaptar la teoría sobre calidad a esta rama de la ingeniería.

Establecer o definir el concepto de calidad como tal no es sencillo, ya que se puede abordar desde diferentes puntos de vista. Comenzando por definirla desde un punto de vista muy general, se puede decir que la calidad es gratis, no se da por sí sola, pero es gratis y significa el grado de conformidad con los requerimientos [[7](#PBC79)]. Esto quiere decir que la calidad significa hacer mejor lo que de cualquier manera se tiene que hacer.

Para esto no se requiere ser demasiado habilidoso ni experto en el tema, sino simplemente poder definirla en términos que no dejen lugar a la interpretación ni a la ambigüedad. Al especificarla de esta manera se vuelve una entidad alcanzable y medible (y por lo tanto mejorable), una vez que se adquiere el compromiso y el entendimiento para realizarla.

Siendo más específicos, la calidad se puede componer de varios significados, pero dos son los más críticos: 1) El desempeño del producto, que termina por relacionarse con el grado de satisfacción del cliente y 2) Que esté libre de defectos, que termina por relacionarse con el grado de insatisfacción del cliente [[8](#JMJ88)]. En cuanto al desempeño, se refiere a la comparación de productos similares dentro del mercado, siendo el objetivo principal el que el producto sea de mayor o igual desempeño que los de la competencia.

Además, los defectos se relacionan con quejas, reclamos, devoluciones y re-trabajo, todas éstas como formas de insatisfacción.

También resulta importante señalar que los defectos pueden impactar a clientes externos o internos. Cuando es el caso de los externos, los defectos representan una amenaza en las ventas futuras. Cuando es el caso de los internos, generalmente constituyen una fuente de aumento en los costos. El objetivo a largo plazo es alcanzar la perfección, es decir, cero defectos. Cabe resaltar que la satisfacción y la insatisfacción no son opuestas, la primera es la razón por la cual se compra un producto, la segunda es la razón por la cual el cliente se queja del producto. Por lo tanto, es posible que un producto esté libre de defectos, pero no sea posible su venta debido a que el desempeño del mismo esté por debajo de la competencia.

Hablando más específicamente de lo que es la calidad en software, el “Institute of Electrical and Electronics Engineers” (IEEE) la define ofreciendo dos alternativas. La primera es:

“*El grado en el que un sistema, componente o proceso cumple con los requerimientos especificados*”.

La segunda es:

“*El grado en el que un sistema, componente o proceso cumple con las necesidades o expectativas del cliente o usuario*”.

La primera alternativa es bastante similar a la primera definición “general” de calidad, y desde el punto de vista del desarrollo de software, esta definición deja de lado los errores que se incluyen al momento de especificar los requerimientos y que por lo tanto no reducen la calidad del software, lo que constituye una deficiencia en el enfoque [[9](#DGa04)]. De igual manera si consideramos la segunda alternativa, y nuevamente desde el punto de vista de desarrollo, esto implica que el cliente esté libre de cualquier responsabilidad en la especificación de los requerimientos, lo que puede traer como consecuencia dificultades en el momento del desarrollo, ya que probablemente los requerimientos no satisfagan las necesidades del cliente [[9](#DGa04)].

Estas definiciones dejan una sensación que la calidad en el software es algo por lo que nos debemos de preocupar una vez que ya esté escrito el código, para comparar el resultado final con lo que se especificó al principio. Sin embargo, si se hace de esta manera nos encontraremos, muy probablemente y muy frecuentemente, con problemas que nos orillarán a emplear una gran cantidad de esfuerzo y dinero en el re-trabajo del producto.

Una definición más completa de lo que significa la calidad en el software es [[10](#RSP92)]:

“*El grado de conformidad con los requerimientos explícitamente establecidos, tanto funcionales y de desempeño, con los estándares de desarrollo explícitamente documentados y con las características implícitas esperadas para todo software desarrollado profesionalmente.*”

Esta definición supone ya una manera de realizar el desarrollo, lo que implica que las actividades de calidad estén presentes en otras fases del proceso igualmente importantes a la codificación y a las pruebas. Es decir, el aseguramiento de la calidad debe abarcar todas y cada una de las fases, desde el análisis y diseño hasta el mantenimiento una vez que el producto salió a producción.

Una vez conociendo esto, se puede afirmar que el aseguramiento de la calidad para un producto específico de software, debe surgir al mismo tiempo que se concibe la idea de desarrollar un nuevo producto, y debe terminar al mismo tiempo que se deja de utilizar el sistema. Sin embargo, todas estas definiciones e ideas expresadas llevan a pensar que el sistema de software tiene que ser, simplemente, perfecto. Por supuesto que el objetivo final, ideal y deseable debe ser éste, siempre y bajo cualquier circunstancia.

Pero así como se han desarrollado estándares de calidad, como el ISO 9001, que establecen los criterios sobre los cuales se puede clasificar un producto o proceso como de calidad o no, así también podemos establecer criterios que definan cuándo un producto de software es de calidad aceptable, es decir, lo suficientemente bueno. Entendiendo por eso que cumple con todos los criterios establecidos para clasificarlo como tal.

* 1. Calidad aceptable

Es indudable que el software perfecto para un sistema complejo no puede ser garantizado en la práctica [[11](#Col94)]. Esto nos puede dirigir hacia otra manera de pensar en la que se aborde el problema desde una perspectiva distinta que nos lleve a definir la construcción de un sistema que sea lo suficientemente bueno, tomando en cuenta situaciones y restricciones presentes en todo proyecto de software, como [[12](#JBa97)]:

* El recurso humano es por mucho el componente más vital y variable en los proyectos.
* Estamos obligados a lidiar con incertidumbre, complejidad, limitaciones, equivocaciones e imperfección en general.
* Todo tiene un costo, y lo que deseamos siempre excede lo que podemos construir.
* En última instancia, la calidad es subjetiva.
* La excelencia no es algo que se alcance mecánicamente, hay que resolver dificultades, valores contradictorios y definir prioridades.
* Teniendo esto en cuenta, es que podemos decir que los métodos para desarrollar software son útiles.

Tomando como premisa estos conceptos, podemos establecer un marco de referencia sobre el cuál evaluar si el producto será lo suficientemente bueno o no. La clave es que este marco de referencia nos ayude a pensar qué es lo importante al momento de evaluar la calidad del producto. En base a esto, el marco de referencia puede contener los siguientes elementos [[12](#JBa97)]:

* El sistema cuenta con los beneficios suficientes, es decir, las partes interesadas se darán cuenta de ellos y contiene todos los beneficios indispensables. Además, el impacto de estos beneficios es el pretendido.
* El sistema no tiene problemas críticos. Qué probabilidad existe que estos problemas se presenten y cuál es su impacto en caso de que lo hagan.
* Los beneficios superan de manera considerable a los problemas.
* Dadas las condiciones actuales, la implementación de mejoras al sistema sería más dañina que benéfica.

Todos y cada uno de estos puntos son fundamentales, por lo que al no cumplir con alguno el sistema podría ser bueno, pero no lo suficientemente bueno. Es importante recalcar que al adoptar este enfoque como tal, no se pretende caer en la mediocridad ni mucho menos en una actitud de “construir el sistema con lo mínimo absolutamente indispensable”. En lugar de esto, lo que se pretende es retroceder un poco y tomar una actitud de pensar y evaluar lo que realmente conviene para el sistema, y no solo realizar las “mejores prácticas” siguiendo al pie de la letra formalidades y requerimientos.

Finalmente, el propósito primordial de este enfoque es crear un nuevo paradigma sobre la idea de calidad en el software, mediante una evaluación crítica de lo que debe contener el sistema para que éste sea exitoso.

* 1. La importancia de la calidad

Sabemos que el costo al momento de realizar cualquier proyecto de software es una de las limitaciones más importantes, sino es que la principal. Esto aplica de manera indistinta para cualquier empresa de desarrollo sin importar su tamaño, **experiencia y madurez**. El objetivo primordial de realizar actividades de calidad es aumentar la probabilidad de que un sistema, al salir a producción, se considere como exitoso.

Sin embargo, es importante tener en mente que siempre que sean bien implementadas, las actividades de calidad permitirán reducir el costo total de desarrollo de un producto en específico al mismo tiempo que se reducen el número de defectos que salen a producción.

* + 1. Costos de la calidad de software

Por eso es importante conocer cuáles son los costos de implementar actividades de calidad y cómo es que se pueden mejorar. Los costos de la calidad se pueden dividir en dos grandes categorías [[9](#DGa04)]:

* Costos de control. Son aquellos que surgen debido a la prevención y detección de errores.
* Costos de fallas de control. Son aquellos que surgen precisamente de la falla en el control.

A su vez, los costos de control se dividen en costos de prevención y en costos de evaluación o revisión. Los costos de prevención incluyen todas aquellas actividades que la empresa realiza de manera general para aumentar la calidad, es decir, actividades como adquisición de infraestructura tecnológica, capacitación a personal y certificaciones. Los costos de evaluación o revisión son aquellos que la empresa realiza para un proyecto en específico. Estos costos incluyen actividades como revisiones técnicas de diseño y código y pruebas de software.

Los costos de falla de control se dividen a su vez en costos internos y externos. Los internos son todos aquellos que se derivan de corregir errores detectados por las actividades de evaluación y revisión. Los externos son aquellos que se derivan de corregir los errores detectados una vez que el producto fue liberado al cliente [[13](#LLa09)]. **La figura 2.1 resume todos estos costos.**

Por supuesto que lo deseable es que todos estos costos se optimicen cada vez que se realiza un nuevo proyecto, inclusive aumentando la calidad al mismo tiempo del producto final. Ahora bien, ¿cómo se puede lograr esto? Sin duda no es tarea fácil, pero definitivamente es algo realizable.

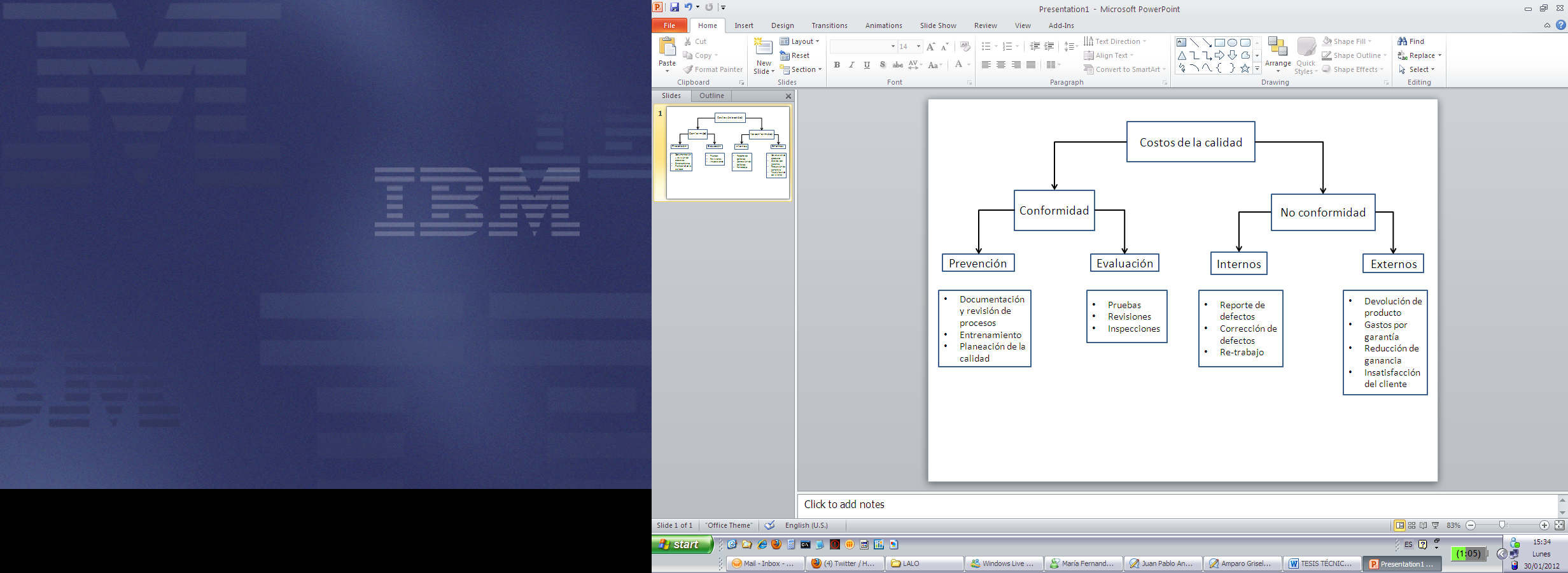


Figura 2.1. Costos de la calidad.

Anteriormente se mencionó que el costo de corregir defectos es mucho menor cuando son detectados en las fases tempranas del proyecto. Es decir, si realizamos actividades de calidad desde el principio, el costo de las fallas internas y externas se reducirá de manera considerable, ya que a final de cuentas se reduce el número de defectos que llegan a producción (los más costosos de corregir). Sin embargo, si realizamos demasiadas tareas de prevención y evaluación puede que sea contraproducente en un punto extremo, ya que los costos de control se pueden elevar de igual manera.

Esto quiere decir que debemos encontrar un equilibrio a medida que se van implementando y haciendo más eficientes las actividades de calidad, pero siempre recordando que es menos costoso implementar calidad en fases tempranas que en tardías. La figura 2.2 [[13](#LLa09)] es una clara muestra de lo que se debe buscar como ideal.

* + 1. Revisiones

Las actividades de calidad en etapas tempranas del desarrollo generalmente consisten en algún tipo de revisión. Ya sea revisión de la especificación de requerimientos, del diseño o del código fuente. En el capítulo 3 se encontrará una explicación detallada de las mismas. Todos estos tipos de revisiones o técnicas de detección tienen el objetivo de encontrar el mayor número de defectos posibles, para evitar que éstos lleguen a fases tardías, como pruebas o producción. Para dar una idea del impacto que tienen estas revisiones, basta repasar el costo de corregir los errores.

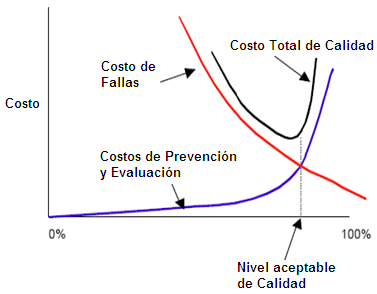


Figura 2.2. Modelo de calidad de software.

Suponiendo que el costo de corregir un error en la fase de diseño es de 1 unidad. El mismo error, pero descubierto antes de que comience la fase de pruebas costará 6.5 unidades; durante la fase de pruebas costará 15, y después de que el producto sea liberado costará entre 60 y 100 unidades [[10](#RSP92)]. Y no solo eso, sino que existe un “efecto multiplicador” que se traduce en un mayor número de errores.

Por ejemplo, suponiendo que existan 10 errores en diseño, que no se realizan revisiones en el proceso, pero que cada etapa posterior corrija el 50% de los defectos al llegar a esa etapa y no introduzca nuevos, esos 10 defectos originales se convertirán en 94 al comenzar la fase de pruebas [[10](#RSP92)].

De acuerdo a una recopilación de datos sobre diferentes proyectos de software mostrada en la tabla 2.1 [[10](#RSP92)], se puede apreciar que el costo total de corrección sin haber realizado revisiones es prácticamente 3 veces mayor al costo habiendo hecho revisiones. Además, el costo de corrección después de haber sido liberado el producto es 4 veces mayor, lo que evidentemente impacta en la calidad final del mismo y en la percepción del cliente, siendo éste uno de los puntos más importantes en las definiciones expuestas de calidad de software.

|  |  |  |  |
| --- | --- | --- | --- |
| **Errores encontrados** | **Número** | **Costo Unitario** | **Total** |
| Con revisiones | | | |
| Diseño | 22 | 1.5 | 33 |
| Antes de pruebas | 36 | 6.5 | 234 |
| Pruebas | 15 | 15 | 315 |
| Producción | 3 | 67 | 201 |
|  |  |  | **783** |
| Sin revisiones | | | |
| Antes de pruebas | 22 | 6.5 | 143 |
| Pruebas | 82 | 15 | 1230 |
| Producción | 12 | 67 | 804 |
|  |  |  | **2177** |

Tabla 2.1 Comparación de costos de calidad.

Finalmente, se puede afirmar que se obtiene un beneficio doble al implementar revisiones a través del ciclo de desarrollo, ya que el costo y el tiempo están íntimamente relacionados. Es decir, menos costo significa menos tiempo de desarrollo y viceversa. En la figura 2.3 se puede observar cómo una reducción de costos a la mitad también se traduce en una reducción del 20% en tiempo, lo que puede traer consecuencias favorables en el mercado y hacia los clientes.

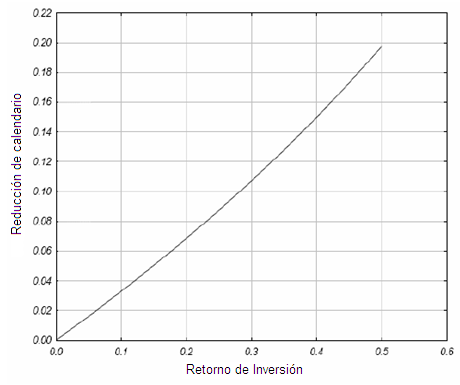


Figura 2.3. Relación costo-calendario.

Con estos datos se puede apreciar de manera contundente la importancia de implementar actividades de calidad durante todo el ciclo de desarrollo del producto, observando que se obtienen beneficios claros y tangibles al reducir el costo total del desarrollo y el tiempo total invertido en el mismo. Lo más importante es que se afectan directa y positivamente dos de las limitaciones más grandes y comunes de cualquier proyecto de software, además de entregar un producto con mayor calidad, lo que sin lugar a dudas el cliente apreciará.

CAPÍTULO 3

1. Implementación de la Calidad

Una vez que se conoce lo que significa la calidad en el software, cuándo un producto puede ser considerado como de calidad aceptable y porqué es importante realizar actividades de calidad, el siguiente paso natural es preguntarse cómo es que se puede implementarla en una empresa para alcanzar los resultados deseados. Los cambios en las empresas siempre causan dolores de cabeza para la empresa misma y sus colaboradores, y definitivamente la implementación de calidad representa un cambio significativo. Por lo tanto, éste debe ser un cambio o proceso que se adopte de manera paulatina e incremental, para evitar la desmotivación por no alcanzar resultados y poder volverse expertos en las tareas realizadas.

Existen muchas maneras de comenzar a implementar actividades de calidad. Sin embargo, considero que son 3 los puntos fundamentales para alcanzar (o por lo menos aumentar significativamente la probabilidad) el éxito:

* Modelos de calidad. En la actualidad existen decenas de modelos que sirven como guías, tratando de facilitar la adopción. Sin embargo, algunos son criticados por la falta de claridad en la definición misma del modelo y los propósitos para los que sirve. Este debe ser un punto de especial atención, para evaluar las ventajas y desventajas de implementar características de los modelos.
* Herramientas de software. Las actividades de calidad representan un esfuerzo considerable, por lo que realizarlas “a mano” consumiría demasiado tiempo. Una herramienta sirve como apoyo para automatizar actividades y dar seguimiento más fácilmente.
* Métricas. Es vital saber si lo que estamos haciendo está bien hecho o no, es decir, si está dando los resultados esperados. Y no hay mejor manera de saberlo que midiendo lo que estamos haciendo. Las métricas proporcionan resultados objetivos y tangibles sobre ello. Además, nos permiten conocer

cómo estamos haciendo las actividades y cómo podríamos mejorarlas cada vez.

En este capítulo se pretende analizar y clasificar los modelos, exponer las condiciones sobres las cuales es preferible utilizar un modelo específico y la explicación de los más conocidos. También se pretende conocer algunas herramientas de apoyo, ya que tiene relevancia con el desarrollo del BM.

* 1. Modelos de calidad

La existencia de modelos que ayuden a definir lo que significa la calidad en productos de software es casi tan antigua como el software mismo. A lo largo de los últimos 30 años se han definido decenas de modelos que pretenden asegurar la calidad de una u otra manera, por lo tanto se volvería impráctico tratar de describirlos todos. A cambio, una descripción de los modelos más conocidos, así como una explicación de la situación actual de éstos, una clasificación de los mismos y una definición de las condiciones o escenarios sobre los cuáles maximizan sus ventajas, ofrece mucho mayor valor.

* + 1. ISO 9126

Este modelo fue desarrollado por la International Organization for Standardization (ISO) por primera vez en 1991 (www.iso.ch), teniendo una revisión y separación en 4 partes a partir del 2001. Prácticamente es mencionado en cualquier texto, revista o publicación acerca de los modelos para calidad de software, por lo que bien vale la pena describirlo.

La primera parte contiene la definición del modelo de calidad. A su vez, este modelo de calidad se divide en dos partes: el modelo de calidad interna y externa y el modelo de calidad en uso. El modelo de calidad interna y externa consiste de 6 características que a su vez se subdividen en 27 sub-características. La figura 3.1 muestra claramente la definición de todas las características y sub-características [[14](#REA09)].

El modelo de calidad interna se centra en las propiedades estáticas de las partes individuales del sistema, como lo son la complejidad y la estructura de los elementos de diseño y codificación. Generalmente estas propiedades se pueden medir durante fases tempranas del ciclo de desarrollo. Estas medidas son significativas por sí mismas.

El modelo de calidad externa compete al momento de ejecutar el software en el hardware, es decir, las propiedades dinámicas son las más importantes. Una medida típica de esto es el tiempo promedio entre fallas, que se relaciona con la confiabilidad dentro del modelo de calidad. Sin embargo, a diferencia de las métricas internas, las externas no tienen significado por si solas, sino que dependen de los elementos externos como el hardware y los datos [[15](#JBo08)].



Figura 3.1. Modelo ISO 9126 para calidad interna y externa.

La segunda parte, el modelo de calidad en uso, establece 4 características que precisamente abordan el uso del sistema [[14](#REA09)]. Estas características son la efectividad, la productividad, la satisfacción y la seguridad, es decir, tiene que ver con las actividades especificadas que realizan los usuarios en un ambiente real.

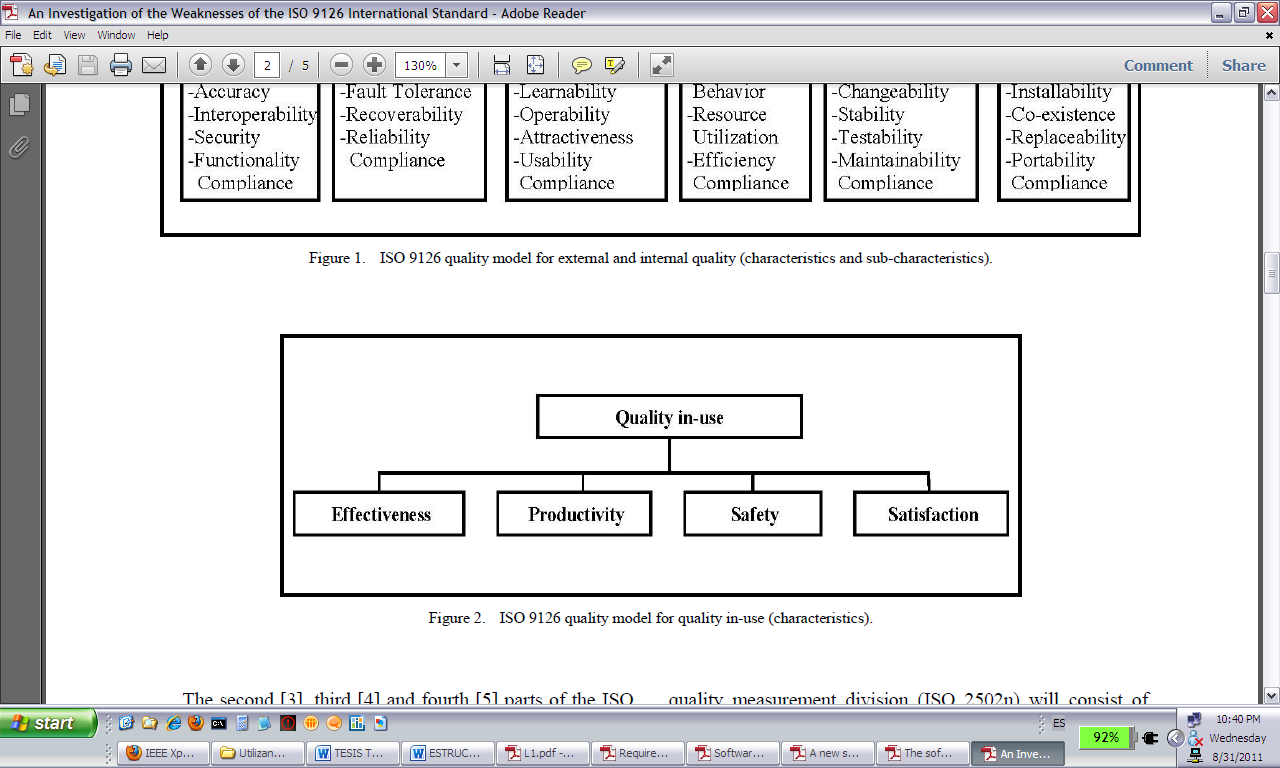


Figura 3.2. Modelo ISO 9126 para calidad en uso.

La segunda, tercera y cuarta parte del estándar proveen información acerca de las métricas externas, internas y de calidad en uso respectivamente. Particularmente se explica el uso y la aplicación de las métricas, proveyendo ejemplos sobre su utilización dentro del ciclo de desarrollo del producto.

Una de las desventajas del modelo es su tamaño. Ya que tratar de definir y medir todas las características y sub-características sería una tarea demasiado grande y compleja, probablemente al grado de superar las tareas propias del desarrollo del sistema. Esto implica que en la gran mayoría de los casos es necesario hacer un análisis de las características más importantes del sistema a desarrollar para enfocar los esfuerzos en la definición y medición de esas características. Probablemente la evaluación para determinar las características más importantes tampoco sea una tarea fácil de realizar, especialmente al inicio del proyecto.

* + 1. CMMI-DEV

De acuerdo al SEI (www.sei.cmu.edu/cmmi), el Capability Maturity Model Integration (CMMI) es un “proceso enfocado a la mejora que provee a las organizaciones de los elementos esenciales de los procesos efectivos que mejorarán su desempeño”. Esto involucra identificar fortalezas y debilidades para convertir éstas últimas en las primeras.

CMMI consiste de una colección de mejores prácticas que se traducen en modelos. Estos modelos a su vez consisten de áreas claves de proceso. Algunas áreas son comunes para todos los modelos y otras son específicas de cada modelo. Los 4 modelos de CMMI son: CMMI-ACQ, CMMI-DEV, CMMI-SVC y People CMM. El modelo de nuestro interés es el CMMI-DEV, ya que es el que se enfoca al desarrollo de productos. De las 16 áreas de proceso comunes, la de aseguramiento de la calidad en el producto y proceso es la que más nos interesa.

Ahora, ¿por qué una empresa en específico decidiría implementar CMMI? Nuevamente de acuerdo al SEI, estas son las ventajas más tangibles que ofrece:

* Se obtienen beneficios con un retorno de inversión significativo.
* Es compatible con estándares y metodologías de desarrollo.
* Es un proceso que está en continua evolución. Este punto es clave, ya que así evita ser un modelo que probablemente se vuelva obsoleto.

Lo interesante acerca de este modelo es que funciona como un marco de referencia en el cuál se definen ciertas características que deben tener los procesos, para considerarlos como de cierta madurez [[1](#PJa00)]. El modelo sólo establece las áreas que se deben cubrir para alcanzar los distintos niveles de madurez, pero no menciona los procesos específicos que se deben seguir. Dicho de otra manera, el modelo especifica el resultado (o el qué) y las áreas de aplicación (o el dónde), pero deja a consideración de cada organización la forma de llegar a ese resultado (o el cómo).

La figura 3.3 [[16](#NEh10)] muestra los niveles en los que se compone el modelo, siendo el nivel 1 el más bajo o el inicial, en el que el proceso prácticamente es impredecible, hasta el nivel 5 u optimizado, en el que se busca la optimización de los procesos continuamente.

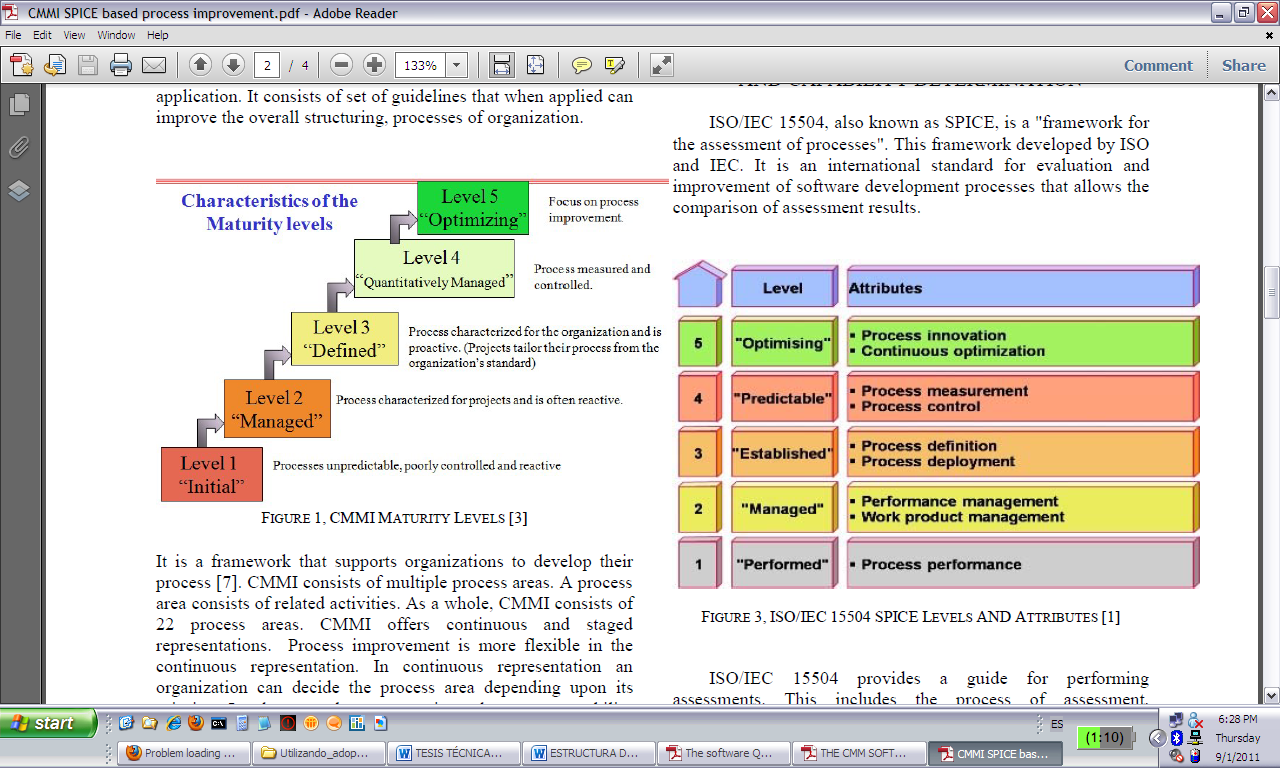


Figura 3.3. Niveles de madurez de CMMI.

Además, para alcanzar cada nivel de madurez se persiguen diferentes áreas de proceso. Sería virtualmente imposible tratar de implementarlas todas de una vez y en ese caso sólo existirían dos niveles. En la tabla 3.1 [[1](#PJa00)] se listan las áreas claves necesarias para alcanzar cada nivel de madurez dentro del modelo CMMI-DEV.

|  |
| --- |
| **NIVEL 2** |
| Administración de requerimientos |
| Planeación de proyectos |
| Seguimiento y revisión de proyectos |
| Administración de personal subcontratado |
| Aseguramiento de la calidad |
| Administración de la configuración |
| **NIVEL 3** |
| Concentración de los procesos de la organización |
| Definición de los procesos de la organización |
| Programa de entrenamiento |
| Administración integrada de software |
| Ingeniería de producto |
| Coordinación intergrupal |
| Revisiones de colegas |
| **NIVEL 4** |
| Administración cuantitativa de procesos |
| Administración cualitativa de software |
| **NIVEL 5** |
| Prevención de defectos |
| Administración de cambios tecnológicos |
| Administración de cambios de procesos |

Tabla 3.1. Áreas de proceso de niveles de CMMI-DEV.

El área de proceso de aseguramiento de calidad se debe enfocar en la planeación de actividades de calidad, verificar la adherencia de los productos y actividades a estándares, procedimientos y requerimientos. Además debe cuidar que las partes interesadas sean informadas de las actividades de aseguramiento de calidad y sus resultados. Por último, debe considerar que los problemas que no puedan ser resueltos durante el desarrollo del proyecto deben ser atendidos por la alta gerencia.

Una de las razones principales por las que este modelo es escogido para asegurar la calidad de los productos, como se mencionó anteriormente, es el beneficio en el retorno de inversión que representa. Este retorno de inversión evidentemente se refleja en la reducción de costos de implementar calidad. La figura 3.4 [[13](#LLa09)] muestra un ejercicio teórico de la reducción de los costos a medida que el modelo de CMMI-DEV se vuelve más maduro en una empresa en específico.

En promedio una empresa de nivel 1 invierte el 60% del costo total de desarrollo sólo en actividades de calidad. Para cuando llega al nivel 3 este costo se reduce al 50%. Una empresa de nivel 5 puede llegar a tener costos menores al 25% del costo total. Cabe señalar también como los costos externos de fallas se reducen más o menos en la misma proporción que los costos totales y cómo aumentan los costos de prevención.

Después de revisar lo que CMMI-DEV contiene, queda una sensación de que es algo demasiado grande y no se visualiza por dónde empezar. Además de que el modelo inmediatamente se asocia con empresas con un gran número de colaboradores. Pero cómo se podría empezar a implementar este modelo en empresas pequeñas, si a final de cuentas ninguna nació grande. H. M. Hosny propone una reestructuración y un reacomodo del área de aseguramiento de la calidad, que facilite su implementación en pequeñas y medianas empresas. Las actividades consisten en [[17](#HMH04)]:

* Iniciar con el desarrollo de planes, estándares y procedimientos que se adecúen fácilmente a la empresa y consideren las particularidades de la misma, facilitando las revisiones durante todo el ciclo de desarrollo del proyecto.
* Realizar un plan de aseguramiento de la calidad práctico para la empresa que se comience a seguir desde el inicio del proyecto, junto con el plan de proyecto.
* Revisar que la propuesta de proyecto esté conforme al objetivo principal y los requerimientos del proyecto.
* Revisar que las actividades del proyecto se realicen conforme al plan, documentando las desviaciones presentadas en el transcurso.
* Revisar que el producto esté conforme a los requerimientos pactados con el cliente.
* Documentar cualquier desviación.
* Definir mediciones de calidad y llevarlas a cabo.

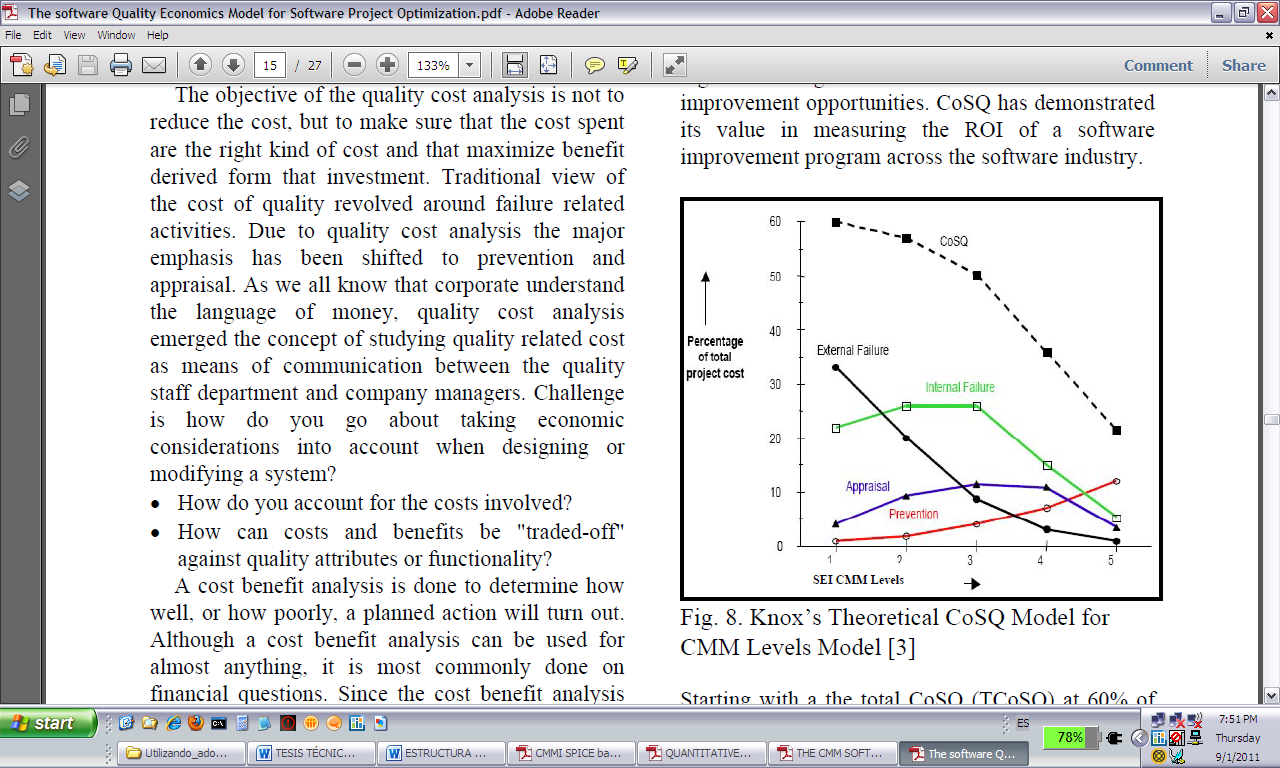


Figura 3.4. Modelo teórico sobre la reducción de costos de calidad.

A final de cuentas, CMMI-DEV es un modelo que proporciona lo que se necesita hacer para asegurar la calidad de productos, específicamente de sistemas de software. Siempre será importante analizar los elementos que se considere puedan agregar valor a la empresa y adaptarlos al contexto actual de la misma.

* + 1. Boehm et al

B.W. Boehm encabezó (por lo menos así se deja entrever) junto con un grupo de personas los primeros estudios que concluyeron en la definición de modelos de calidad para los sistemas de software. Por eso considero importante el que se conozca su modelo. Curiosamente, existieron estudios previos que se pueden considerar como un esbozo de los primeros modelos de calidad. Específicamente nombra a los autores R. J. Rubey y R. D. Hartwick como los primeros en desarrollar métodos para evaluar la calidad del software mediante la definición de atributos y métricas en el año de 1968 [[18](#BWB76)].

La razón principal de sus estudios fue precisamente abordar varios atributos de calidad difíciles de definir y evaluar en aquella época (algunos se mantienen hasta el día de hoy), con el fin de crear una estructura que permitiera evaluar cuantitativamente la calidad de un producto. Particularmente buscaba que [[18](#BWB76)]:

* Dado un programa específico, comprobar que lo desarrollado iba acorde con lo especificado mediante una o más métricas.
* La calidad global del software fuera el resultado de aplicar una función al valor de las métricas.

Todo esto lo hizo pensando específicamente en la evaluación de programas desarrollados en Fortran. De los estudios realizados obtuvieron como conclusiones principales que cualquier métrica, por más simple que fuera, siempre contenía un lado opuesto que comprometía su credibilidad. También concluyeron que el software evoluciona tan rápido, que establecer métricas podría resultar contraproducente en casos específicos.

Además determinaron que calcular una métrica para evaluar la calidad global traería más problemas que beneficios, debido a que las características individuales de la calidad algunas veces entran en conflicto. Al analizar estas situaciones, determinaron que lo mejor era:

* Desarrollar una serie de características que fueran importantes para el sistema, que no se traslaparan y estuvieran completas.
* Definir métricas que sirvieran como guías para características individuales.
* Determinar la correlación de las métricas y características con la calidad, así como su impacto.
* Darle la importancia a cada métrica respecto al punto anterior.
* Finalmente, mejorar las métricas y características para conformar un grupo lo más mutuamente exclusivo posible y completo para evaluar la calidad.

En base a esto, desarrollaron una serie de características que consideraron como iniciales. Después, al revisar las particularidades del lenguaje Fortran y desarrollar mediciones para éste, se percataron que las mediciones aplicaban para más de alguna característica, por lo que identificaron que las características se relacionaban en una especie de árbol.

Así fue como desarrollaron un segundo nivel de características. Al profundizar en su estudio, encontraron que todas se podían representar en el árbol, quedando las iniciales como las características más específicas (en el último nivel del árbol) y a partir de éstas establecieron unas más generales. La figura 3.5 [[18](#BWB76)] muestra el árbol original de características.

Posteriormente desarrollaron una serie de métricas que correspondían a una característica primitiva y las evaluaron en 5 aspectos mediante tablas de rangos y criterios para cada aspecto. Así cada métrica era evaluada en su potencial benéfico, su facilidad de ser cuantificada, su facilidad de ser calculada automáticamente, qué tan completo era el cálculo y su correlación con la calidad. Finalmente utilizaron las características de calidad y sus métricas para mejorar gradualmente el proceso de desarrollo.

Así fue como surgió probablemente el primer modelo de calidad completo y similar a los desarrollados hoy en día. Podemos observar que la mayoría de las características se utilizan hasta el momento y que también se utiliza un árbol jerárquico para determinar su relación. También se establece el primer esfuerzo considerable por cuantificar la calidad a través de métricas.

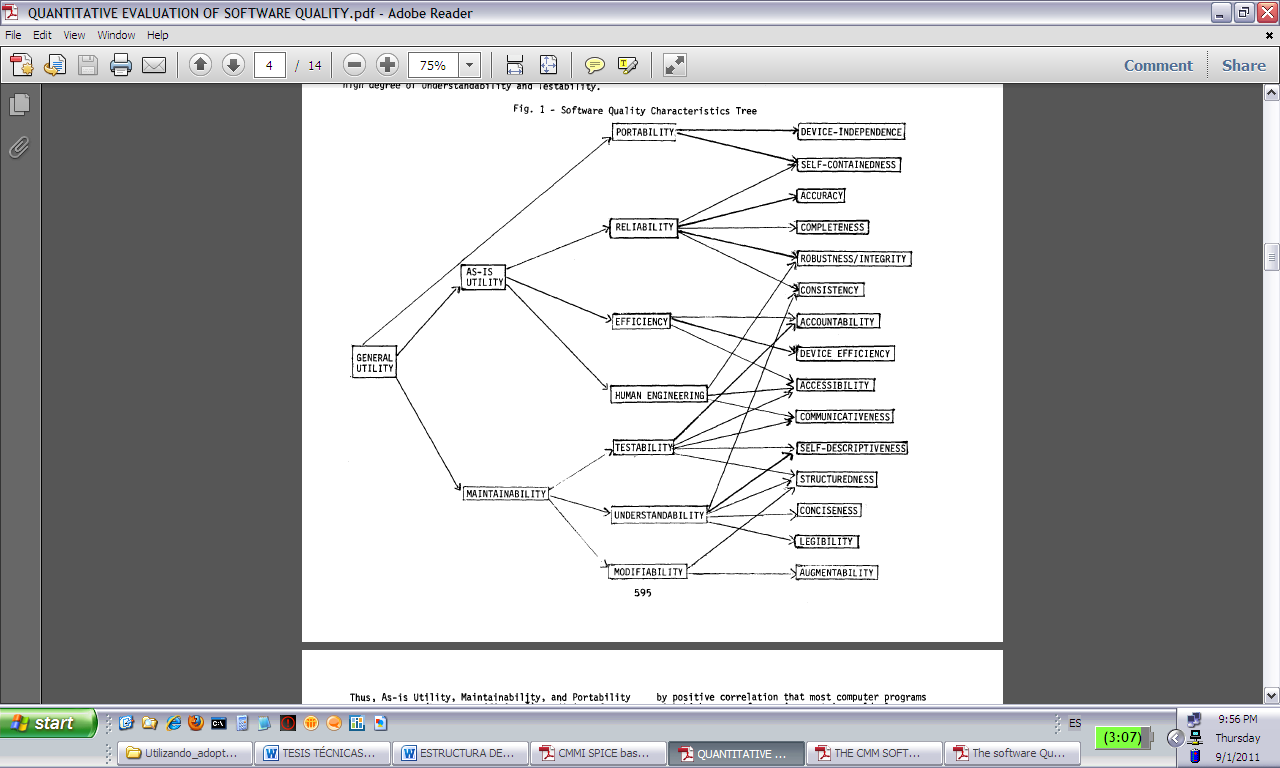


Figura 3.5. Árbol de características de calidad de software.

* + 1. Clasificación, situación actual y escenarios de los modelos de calidad

Después de haber revisado 3 modelos de los más conocidos dentro de la industria, es importante mencionar la situación actual de los mismos, analizando el contexto en el que pueden ofrecer más ventajas si son implementados, así como los escenarios en los cuales su implementación no significaría una mejora considerable en la calidad final.

Como se comentó anteriormente, en los últimos 30 años se han desarrollado una gran variedad de modelos con el objetivo de mejorar la calidad al momento de desarrollar software. Todos estos modelos han conseguido diferentes resultados cuando son aplicados. Sin embargo, existe la sensación de que a pesar de los esfuerzos realizados por estandarizar de alguna manera estos modelos, todavía falta claridad en su definición y en los objetivos que persiguen.

Con esto en mente, vale la pena hacer un análisis de los modelos existentes estableciendo propósitos y escenarios en los que funcionan mejor, así como una guía de requerimientos generales para estos modelos que eventualmente lleven a una mejora en su definición e implementación.

De la gran variedad de modelos, que en sí ha constituido más un problema que una solución, hay algunos que apoyan en la especificación de requerimientos de calidad para el sistema, otros que evalúan los sistemas utilizando métricas y unos más que tratan de predecir la calidad de la versión actual del sistema basados en modelos de regresión.

Teniendo en cuenta esto, se puede desarrollar una clasificación de los modelos que precisamente se relacione con el propósito principal del modelo, es decir, podemos clasificar entonces los modelos en los que ayudan a definir la calidad, los que ayudan a evaluar la calidad, y los que ayudan a predecir la calidad [[19](#Dei)]. Aun cuando abordan diferentes propósitos, éstos no son independientes. Sería difícil evaluar la calidad sin saber que es, y más complicado aún predecirla sin saber cómo evaluarla o medirla. En base a esto se puede concluir que los modelos de predicción son los más completos, ya que se requiere definir para medir y medir para predecir. La figura 3.6 [[19](#Dei)] muestra la relación y clasificación de los distintos modelos.

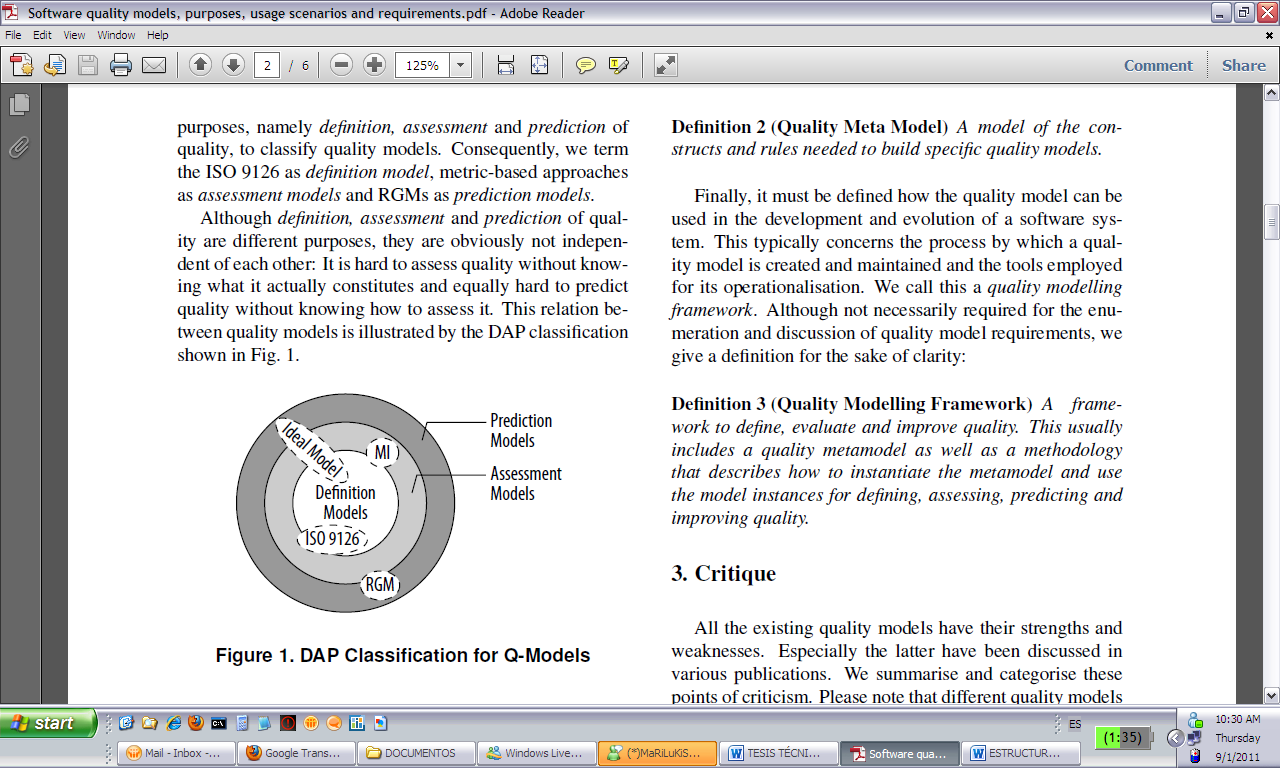


Figura 3.6. Clasificación de modelos de calidad de software.

Ya que se establece una clasificación, se puede hacer un mejor análisis de las áreas de oportunidad de cada tipo de modelo, para posteriormente conocer los escenarios en los que sí contribuyan significativamente y los requerimientos mínimos de cada tipo que permitan evaluarlos y mejorarlos. Se pueden identificar como debilidades generales de los modelos las siguientes [[19](#Dei)]:

* La falta de integración en las actividades de calidad. Por ejemplo, la especificación de requerimientos de calidad y la evaluación de la calidad del sistema no se basan en los mismos modelos.
* Los modelos actuales no abordan el problema de la calidad desde diferentes perspectivas. No es lo mismo abordarlo desde la perspectiva del cliente que de la perspectiva de ingeniería.
* Los modelos no toman en cuenta las diferencias inherentes a los sistemas de software, es decir, no es lo mismo realizar un sistema de inteligencia de negocios que un controlador embebido.

Dentro de los modelos clasificados como de definición, la principal debilidad reside en la falta de claridad para establecer criterios de descomposición de los atributos del sistema. Estos criterios confían demasiado en la taxonomía de la definición y no siguen guías para su descomposición. Esto causa con frecuencia que los atributos se traslapen.

Para el caso de los modelos de evaluación, el que gran parte de los atributos sean demasiado abstractos para ser medidos directamente, causa que su implementación sea difícil para la gran mayoría de proyectos. Así mismo, los modelos de predicción terminan definiendo modelos matemáticos que resultan complejos de interpretar y utilizar. Al conocer las dificultades o debilidades de los distintos tipos de modelos, podemos establecer escenarios y condiciones bajo los cuales su implementación puede resultar atractiva.

Por ejemplo, los modelos de definición apoyan en la especificación de requerimientos de calidad, por lo que pueden ser una excelente referencia para acordar la calidad con el cliente. Además, pueden servir como base para la definición de estándares de diseño y codificación, ya que proveen recomendaciones directas para el desarrollo de sistemas. Por último, sirven como una buena guía para clasificar los defectos inyectados durante el desarrollo.

Los modelos de evaluación facilitan la especificación objetiva de los requerimientos de calidad, son la base de las mediciones de las actividades de calidad y sirven como guías para la definición de revisiones y uso de herramientas de análisis estático. Los modelos de predicción funcionan principalmente en la administración del proyecto, ya que se pueden utilizar para establecer el esfuerzo de desarrollo y pruebas del sistema.

Finalmente, conociendo las debilidades y condiciones bajo las cuales los modelos maximizan su contribución, se pueden definir requerimientos generales y específicos para el desarrollo de nuevos modelos y para la mejora de modelos actuales. Algunos de estos requerimientos son [[19](#Dei)]:

* Todo modelo de calidad debe especificar cómo puede ser implementado dentro del ciclo de desarrollo.
* Los modelos de definición deberían basarse en un Meta modelo explícito. Este Meta modelo debería contener la estructura de descomposición del modelo sin admisión de ambigüedades, lo que permitiría definir los criterios de calidad más fácilmente con el cliente.
* Los modelos de evaluación deben contener criterios que se puedan describir con métricas. Si existiera algún criterio que no se pudiera describir así, al menos debería poder ser descrito cualitativamente sin traslaparse con otros atributos.
* Los modelos de predicción deben desarrollar modelos matemáticos comprensibles y fáciles de interpretar, con predicciones sobre la fase de pruebas y los ciclos de liberación de nuevas versiones.

La decisión de adoptar un modelo o modelos de calidad para el desarrollo de sistemas de software dentro de una empresa no debe ser algo que se tome a la ligera. Por eso es importante conocer cuáles modelos pueden servir más en las condiciones en las que se encuentra la empresa.

Además de tener siempre en mente que no es obligatorio seguir al pie de la letra estos modelos, ya que finalmente sirven como guías. En su lugar, evaluar a conciencia las actividades de los modelos que ayudarían a la empresa y las actividades que no lo harían tanto, para tratar de maximizar el efecto positivo de incorporar nuevas actividades al desarrollo de sistemas.

* 1. Herramientas de software

Además de contar con guías sobre cómo incrementar y asegurar la calidad de los sistemas de software, también es importante contar con herramientas que permitan automatizar tareas, producir métricas importantes y dar un mejor seguimiento al avance de los proyectos y los defectos que van surgiendo en el camino. Es aquí donde los sistemas de seguimiento encajan dentro del ciclo de desarrollo.

Al igual que con los modelos de calidad, se pretende presentar una descripción de los principales sistemas, así como una evaluación de la situación actual de los mismos, exponiendo sus funcionalidades. Esto con el objetivo de conocer el panorama general de estas herramientas y tener una idea clara del contexto sobre el cual trabajan, para determinar dónde puede encajar el BM, que es a final de cuentas el sistema a desarrollar y uno de los dos objetivos principales de este trabajo de investigación.

Es importante señalar que existen diferentes tipos de herramientas o sistemas, con diferentes tamaños y alcances y con distintos tiempos de implementación. Existen desde las herramientas que solamente se especializan en el registro y seguimientos de defectos hasta las que involucran la administración de versiones junto con la gestión del proyecto. Bien se podría realizar todo un trabajo de investigación sobre los diferentes tipos de herramientas, analizando las fortalezas y debilidades de cada una, así como evaluando cuándo y bajo qué circunstancias estas herramientas maximizan su beneficio.

En base a esto, podemos entonces clasificar las herramientas como de 4 tipos:

* Sistemas de rastreo de defectos.
* Sistemas de control de versiones
* Sistemas de gestión de proyectos.
* Sistemas que combinan 2 o más de estas funcionalidades.

Los sistemas de control de versiones se especializan en el registro y control de los cambios en las versiones del producto en desarrollo. Algunas de estas herramientas manejan “líneas base”, es decir, versiones de código fuente que generalmente se asocian con la liberación del producto al mercado. La principal ventaja radica en poder “navegar” entre las diferentes versiones para en un momento dado regresar a una versión anterior con alguna funcionalidad específica o en el caso de que a la versión actual se le hayan hecho cambios no deseados a final de cuentas.

Fácilmente existen más de 20 herramientas, unas con costo y otras sin costo, que facilitan la administración de las versiones del producto con diferentes funcionalidades entre ellas. La tabla 3.2 muestra las herramientas que considero como las más conocidas.

|  |  |
| --- | --- |
| **Nombre** | **Licencia** |
| Rational Clear Case | Propietaria |
| CVS | GPL |
| CVSNT | GPL o propietaria |
| Dimensions | Propietaria |
| Rational Team Concert | Propietaria |
| Subversion (SVN) | Apache BSD |
| Visual SourceSafe | Propietaria |

Tabla 3.2. Sistemas de control de versiones.

Los sistemas de rastreo de defectos probablemente sean los más conocidos y adoptados dentro de la industria. Ganaron popularidad al crecer el número de integrantes dentro de los proyectos, además de estar establecidos en localidades remotas. Esto provocó que en la actualidad existan más de 70 herramientas (<http://www.aptest.com/bugtrack.html>) que faciliten el registro, clasificación, búsqueda y seguimiento de defectos en diferentes proyectos de software.

Igual que los sistemas de control de versiones, algunas de estas herramientas son de uso libre, mientras que otras cuentan con diferentes esquemas de licenciamiento. Dentro de las herramientas más conocidas se encuentran:

* BugAware.com
* BUGtrack.
* Bug Tracker Software.
* Bugzero
* Bugzilla
* JIRA
* Defect Manager
* SWBTracker
* ZeroDefect

Algunas de estas herramientas son independientes de la plataforma, con posibilidad de utilizar varias bases de datos para almacenamiento de información. La mayoría tienen un ambiente web. Algunas cuentan con la capacidad de registrar defectos desde el correo electrónico. Es importante tomar en cuenta ciertos aspectos al momento de comenzar a utilizar una herramienta. Estos sistemas (y creo que ninguno) por sí solos no resolverán los problemas de registro y seguimiento de defectos que pueda tener la empresa.

El uso que se le dé a la herramienta será vital para alcanzar los objetivos deseados al momento de utilizarla. Algunas recomendaciones importantes son [[20](#NSe05)]:

* El registro correcto de los defectos es lo más importante, incluso por encima de la herramienta misma.
* El diseño de la aplicación es básico, ya que permitirá identificar los defectos de manera rápida.
* Decidir el alcance de uso para la herramienta es importante. Es diferente utilizarla como registro de tareas a rastreo de defectos.
* Definir un pequeño entrenamiento comunicando la manera en que se registrarán los defectos, esto llevará a un mejor entendimiento los registros.
* No utilizar la herramienta como seguimiento puntual al trabajo de los miembros del equipo. Si se sienten vigilados, introducirán datos falsos, lo que volverá inútil la herramienta.

Los sistemas de gestión de proyectos son los más completos por sí solos, ya que su propósito se extiende a lo largo de todo el ciclo de desarrollo. Los sistemas más exhaustivos incluyen la planeación y la estimación del esfuerzo del proyecto, la posibilidad de establecer un calendario de desarrollo, la administración de los costos y presupuestos, la asignación de recursos y el manejo de la documentación generada.

Al igual que con los sistemas anteriores, existen una gran cantidad de sistemas disponibles con diferentes funcionalidades, pero con el mismo propósito de administrar y gestionar correctamente los proyectos de software. Entre los más conocidos podríamos mencionar los de Microsoft, como Microsoft Project, Microsoft SharePoint y Microsoft Team Foundation Server. También la herramienta de HP (Project & Portfolio Software), la de Oracle (Project Portfolio Management) y algunas otras más.

Como podemos concluir, entre los 3 tipos de herramientas, probablemente existen más de 200 herramientas disponibles para diferentes empresas con una gran variedad de necesidades específicas. Inclusive algunas de estas herramientas pueden caer en el cuarto tipo mencionado anteriormente, ya que la combinación de sus funcionalidades permite atacar varios propósitos. A continuación se presentan un poco más a detalle las funcionalidades de 3 herramientas conocidas.

* + 1. Bugzilla

Probablemente esta sea la herramienta más famosa actualmente, lo que no necesariamente la hace la mejor. Precisamente surgió a partir de la necesidad de contar con una herramienta de rastreo de defectos para el desarrollo del navegador “Mozilla” [[20](#NSe05)]. Está desarrollado en el lenguaje programación PERL y funciona con la base de datos MySQL. A su vez, el servidor web sobre el que está montado es Apache. Dentro de las principales características del sistema se encuentran ([www.bugzilla.org/features](http://www.bugzilla.org/features)):

* Notificaciones vía correo electrónico configurados a través de las preferencias de usuario.
* Diferentes vistas con información distinta de los defectos.
* Reportes que pueden ser enviados vía correo electrónico de manera automática.
* Gráficas y reportes sobre el estado de los defectos dentro de un proyecto.
* Algoritmos de reconocimiento de defectos similares, para evitar defectos duplicados.
* Sistema avanzado de búsqueda de defectos.

Esta última podría ser la característica más marcada del sistema, debido a que el motor utilizado permite realizar búsquedas bastante complejas y flexibles. Sin embargo, y especialmente para nuevos usuarios, puede ser complicada de utilizar, ya que existen criterios como “severidad” o “prioridad” que no quedan totalmente claros [[21](#NJo03)]. Resultaría interesante contar con una versión más sencilla del motor para realizar algunas búsquedas. Contrario a esto, la pantalla que muestra los resultados de la búsqueda resulta bastante sencilla y mucho más fácil de interpretar. Además, la información mostrada en esta pantalla representa una buena síntesis. A partir de aquí se puede seleccionar cada defecto para ver la información completa al respecto.



Figura 3.7. Búsqueda de defectos en Bugzilla.

* + 1. GanttProject

Es un sistema de administración y calendarización de proyectos, similar a Microsoft Project, inclusive en su interfaz de usuario. Al estar desarrollado en Java, puede ser utilizado en diversos sistemas operativos como Windows, Linux o Mac OS. Opera bajo la licencia GPL. Las principales funcionalidades con las que cuenta son [[22](#Gan)]:

* Creación de gráficas de Gantt con información de las tareas y sus dependencias, así como sus hitos.
* Todas las tareas se organizan en una estructura de árbol, por lo que se puede definir una estructura lógica de división del trabajo.
* Es posible asignar los recursos definidos a cada una de las tareas del proyecto.
* Permite exportar las gráficas en formatos PDF, HTML y PNG.
* Además es posible interactuar con Microsoft Project a través de la importación de proyectos en ese formato y la exportación de los creados en GanttProject a ese formato. Además, es posible generar archivo separados por coma (CSV, por sus siglas en inglés).
* Por último, es posible compartir proyectos a través de WebDAV, que básicamente es un servidor web en el que se pueden administrar y editar archivos.

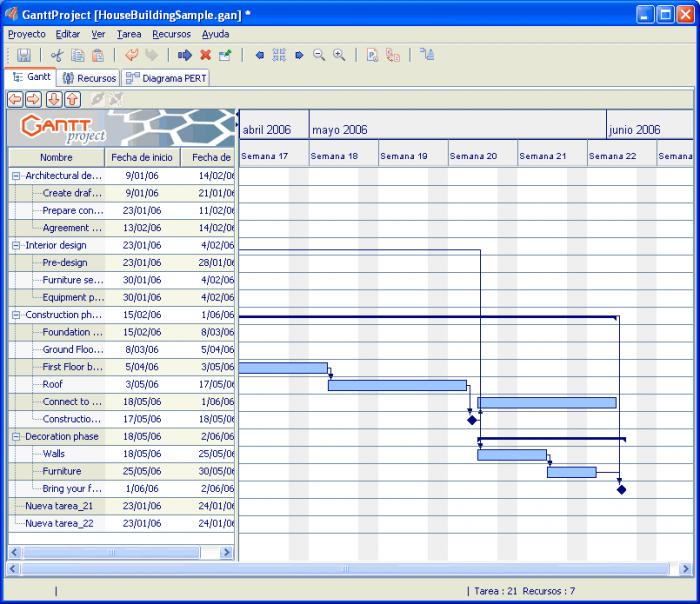


Figura 3.8. GanttProject.

* + 1. Rational Clear Quest

Es una herramienta de apoyo para la gestión de cambios y de rastreo de defectos. La herramienta es desarrollada y mantenida por International Business Machines (IBM). Puede ser utilizada con la configuración de instalación y modificarse a las necesidades de la empresa y del proyecto. La herramienta ayuda a la gestión de proyectos mediante el manejo del flujo de trabajo de los miembros del equipo. Además, se pueden visualizar los elementos afectados por las solicitudes de cambio, ya sean mejoras o correcciones. Las principales funcionalidades y características de la herramienta son [[23](#IBM)]:

* La configuración y automatización del flujo de trabajo, permitiendo crear procesos repetibles y predecibles.
* Control de acceso con un proceso de identificación electrónico que permite asegurar que las solicitudes de cambio y los cambios fueron realizados por las personas adecuadas.
* Cálculo de métricas y realización de reportes en tiempo real.
* Administración de los cambios en el proyecto a través de las actividades realizadas, lo que permite mayor claridad al momento de comparar cambios.
* Plantillas para la gestión del ciclo de vida de la aplicación.
* Soporta desde equipos pequeños de desarrollo hasta equipos distribuidos remotamente.

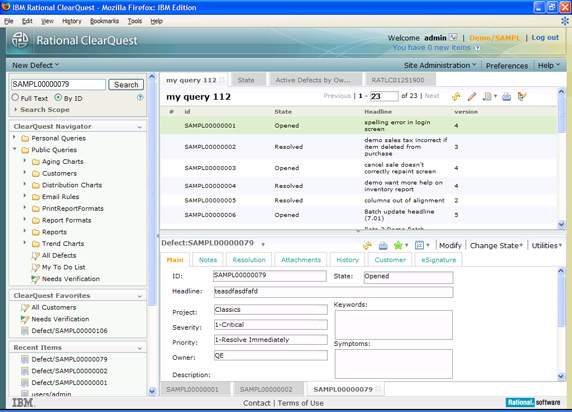


Figura 3.9. Rational Clear Quest.

Una vez que se conoce la cantidad de herramientas existentes y las principales funcionalidades con las que cuentan, nos damos cuenta que al realizar un nuevo sistema no estamos descubriendo el hilo negro ni mucho menos. Por lo tanto el objetivo se vuelve aportar una nueva funcionalidad al momento de desarrollar un nuevo sistema o facilitar aún más la manera en que se realizan ciertas actividades correspondientes al ciclo de desarrollo. También una determinada combinación de funcionalidades específicas puede provocar que el propósito final del sistema sea de valor. Esto último es lo que se busca alcanzar con el desarrollo del BM, ya que se pretende que las empresas que no tienen implementadas actividades de calidad, especialmente pequeñas y medianas, comiencen a realizarlas sin que esto represente un cambio demasiado brusco en la manera de desarrollar sus proyectos.

CAPÍTULO 4

1. Conociendo la Calidad

Si bien medir lo que se hace referente a la calidad es parte de su implementación, junto con la adopción de modelos de calidad y el uso de herramientas de software, su principal beneficio radica en que nos permite conocer si lo realizado hasta el momento ha traído beneficios significativos o si es necesario un reajuste a las actividades. Al mismo tiempo, permite conocer si la calidad del producto es aceptable o no. Dicho de otra manera, existen métricas que nos permitirán evaluar la calidad final del producto en su versión inicial y posterior mantenimiento, y existen métricas que nos permitirán determinar si el proceso para desarrollar el producto es de calidad o no. Por ende, podemos dividir las métricas de calidad en métricas de proceso y métricas de producto [[24](#SHK95)].

Una métrica en el contexto de calidad de software significa una función en la cual sus entradas son mediciones básicas de software y su salida un valor que representa el grado en el que el software posee determinado atributo [[25](#NFS97)].

* 1. Métricas de producto

Las métricas de producto son aquellas que nos permitirán evaluar la calidad final del mismo, siendo las dos principales el tiempo promedio entre fallas y la densidad de defectos. El tiempo promedio entre fallas se utiliza principalmente en sistemas de software críticos, como el sistema para controlar un avión o el tráfico aéreo. La densidad de defectos es una métrica mucho más conocida y utilizada, ya que es la principal para cualquier software comercial, es por eso que nos enfocaremos en esta última.

La densidad de defectos significa esencialmente el número de defectos que contendrá el sistema por unidad de tamaño. De primera instancia, su definición parece bastante simple y no implica mayor esfuerzo. Sin embargo, hay varios detalles que se deben considerar al momento de su cálculo para obtener valores de utilidad. Primero que nada se debe de definir la unidad de tamaño a utilizar, las dos más comunes son

las líneas de código **(LOC, por sus siglas en inglés) y puntos de función (FP, por sus siglas en inglés)**. Uno podría pensar que al escoger como unidad las líneas de código no debería haber mayor problema para calcular la densidad. ¿Y qué significa contar las líneas de código? En realidad es algo importante al momento del cálculo, ya que hay varios puntos que se deben tomar en cuenta para determinar el número total de LOC:

* Contar el número físico de líneas.
* Contar el número de líneas ejecutables.
* Contar el número de instrucciones.
* Contar el número de líneas ejecutables más las declaraciones.
* Contar el número de líneas ejecutables más los comentarios y las declaraciones.
* Lenguaje de programación. Varía el número de líneas físicas para formar una instrucción.

Así puede haber muchas combinaciones más, y cada una de ellas traerá diferentes resultados. Lo importante es definir un estándar y ser consistente con ello. A su vez, los FP surgieron como idea para tratar de mitigar las desventajas que representaba no contar con un estándar definido para contar las LOC. Esencialmente, los FP son un conjunto de instrucciones que realizan una tarea en específico, junto con las declaraciones y la manipulación de las variables locales por esas instrucciones [[24](#SHK95)].

Conociendo esto, se podría pensar que se eliminan las dificultades de determinar las LOC, debido a las diferencias en cuanto a la manera de contarlas y a los diferentes lenguajes de programación. Estas dificultades se resuelven, pero se agregan unas diferentes. Por un lado se tienen que determinar factores de acuerdo a la complejidad de las funciones. Por otro lado, se tiene que hacer una evaluación (en una escala del 0 al 5) del impacto de algunas características generales del sistema. Finalmente se calcula un valor de ajuste para determinar el número final de FP.

Es decir, cualquiera de las dos unidades conlleva elementos que se pueden considerar como subjetivos, o que por lo menos conducen a resultados distintos, por lo que es tarea de los miembros del equipo y de la empresa determinar la mejor manera de calcular la densidad final de defectos. No solamente es importante calcularla para la versión inicial del sistema, sino que se debe calcular con cada versión posterior del mismo. Esto agrega dificultad al evaluar la calidad del sistema, ya que involucra el cálculo de la densidad de defectos introducido en los cambios de la versión previa a la actual. Además, si consideramos que el objetivo con nuevas versiones es aumentar la calidad, entonces debemos tomar en cuenta que se necesita reducir cada vez la densidad de defectos.

Debido a que la densidad está directamente relacionada con el tamaño del sistema, si una versión es mucho mayor que sus anteriores, esto quiere decir que la densidad tendrá que ser considerablemente menor para reducir el número total de defectos en relación a la versión previa [[24](#SHK95)]. Como podemos observar, el cálculo de la densidad de defectos deja de ser trivial al momento de considerar todas estas situaciones, por lo que es importante prestarle atención para obtener los resultados más exactos posibles.

* 1. Métricas de proceso

Ya sabemos que las métricas de producto son una manera cuantificable de determinar la calidad final de los sistemas de software. ¿Pero qué significa que un producto sea de calidad? ¿Cuál debe ser su densidad final para determinar que el producto sea de calidad? La tabla 4.1 [[26](#WSH09)] muestra la densidad de defectos por nivel de madurez en el modelo CMM.

|  |  |  |
| --- | --- | --- |
| **Nivel CMM** | **Defectos/KLOC** | **Defectos/MLOC** |
| 1 | 7.5 | 7500 |
| 2 | 6.24 | 6240 |
| 3 | 4.73 | 4730 |
| 4 | 2.28 | 2280 |
| 5 | 1.05 | 1050 |

Tabla 4.1. Densidad de defectos por nivel de CMMI.

De acuerdo a las densidades, esto significa que un producto con un millón de líneas tendría más de 1000 defectos. Y de acuerdo a los sistemas que se realizan hoy en día no sería descabellado pensar que una gran parte de ellos fuera de este tamaño. ¿Sería este un programa en el que confiaríamos? Si pensáramos en los sistemas operativos, ahora cualquier sistema contiene fácilmente más de 10 millones de líneas. Si todos se desarrollaran con un nivel 5 de madurez, esto significa que en promedio tendrían más de 10,000 defectos.

Por eso es importante reflexionar acerca de la calidad con la que deben salir al mercado los sistemas de software. Si no realizáramos actividades de calidad y todo se lo dejáramos a las fases de pruebas, ¿**Cuál sería** el costo y el impacto que tendría en el producto? Todo el proceso que se requiere para corregir un defecto que se haya escapado y que haya sido detectado por un usuario final. Debido a todas estas actividades para corregir 1 sólo defecto, es que el costo de hacerlo en esta etapa es considerablemente mayor que en etapas tempranas del proyecto. Para mejorar la calidad del producto, es indispensable mejorar la calidad del proceso [[26](#WSH09)], y para saber si el proceso es de calidad, necesitamos medir lo que hacemos y producir métricas de valor.

Ahora, ¿cuáles métricas permiten conocer lo que estamos haciendo? Se deben definir métricas que evalúen el trabajo y no solo los productos de trabajo. Por eso se necesitan métricas sobre el rendimiento de detección de defectos, el costo de la calidad, las tasas de revisión, las tasas de fases y el índice de la calidad de proceso [[26](#WSH09)].

* + 1. Rendimiento del filtro de remoción de defectos

El rendimiento es una métrica que nos indica la eficiencia de una etapa para detectar o remover defectos. Se ha hablado acerca de implementar actividades de calidad a lo largo de todo el ciclo de desarrollo, por lo que cada actividad se puede considerar como un filtro con el propósito de remover tantos defectos como sea posible. Al momento de realizar el cálculo, también es necesario tomar en cuenta los defectos que se hayan inyectado y removido en esa etapa. Por lo que el **rendimiento del filtro** se puede definir como

|  |  |  |
| --- | --- | --- |
|  |  | (4.1) |

Es posible medir el rendimiento para cada etapa del proyecto, es decir, se puede tener un rendimiento de requerimientos, de diseño, de codificación y de cualquier otra etapa. También es común calcular el rendimiento del proceso, que significa el rendimiento en defectos removidos hasta antes de la etapa de compilación, en caso de que se tenga una, en caso contrario hasta antes de las pruebas unitarias. La figura 4.1 [[26](#WSH09)] muestra el avance en rendimiento de 12 desarrolladores a lo largo de 10 implementaciones utilizando la metodología Proceso Personal de Software (PSP, por sus siglas en inglés). En promedio, al no implementar actividades de calidad se tiene un rendimiento casi del 10%, al implementar las actividades sugeridas por la metodología PSP se llega a tener un rendimiento prácticamente del 70%.

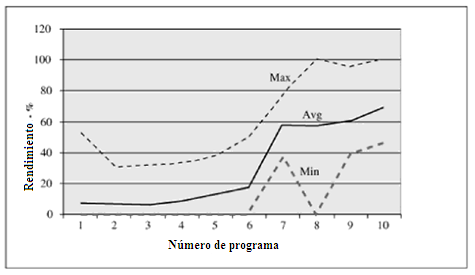


Figura 4.1. Rendimiento de proceso

* + 1. Costo de la calidad

Anteriormente se mencionaron los costos de implementar calidad en el ciclo de desarrollo. Por simplicidad, estos costos se resumen en costos de fallas y costos de evaluación. Por lo que el . Los costos de fallas se consideran como los costos de realizar pruebas y compilación. Los costos de evaluación se consideran todos aquellos realizados antes de la etapa de compilación o pruebas, es decir, todas las actividades de revisión hasta ese momento.

Por lo tanto, el costo de las fallas es

|  |  |  |
| --- | --- | --- |
|  |  | (4.2) |
|  |  |  |
|  |  | (4.3) |

A su vez, el costo de evaluación es

|  |  |  |
| --- | --- | --- |
|  |  | (4.4) |

De estas mediciones se pueden derivar dos métricas importantes. La primera es el porcentaje que nos significa realizar actividades de evaluación sobre el costo total y la segunda y más relevante, la relación costo evaluación-costo fallas (A/FR). Por lo que el

|  |  |  |
| --- | --- | --- |
|  |  | (4.5) |

|  |  |  |
| --- | --- | --- |
|  |  | (4.6) |

Una relación de A/FR mayor a 1 significa que estamos invirtiendo el mismo tiempo en revisar que en corregir defectos en pruebas. **Entre** mayor sea esta relación menor será el número de defectos [[26](#WSH09)]. **Un A/FR mayor a 2 es adecuado.** **Sin embargo**, una relación demasiado grande significa que estamos pasando mucho tiempo en revisar, lo que probablemente tampoco sea deseable. La figura 4.2 [[26](#WSH09)] la relación A/FR de los mismos 12 desarrolladores de la figura 4.1 en relación al número de defectos por cada mil líneas de código encontrados en pruebas unitarias.

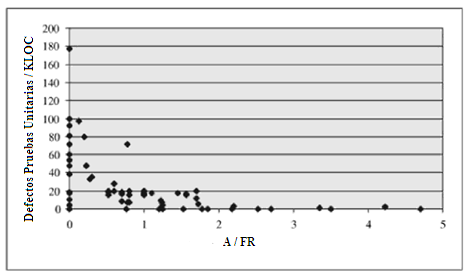


Figura 4.2. Relación A/FR y defectos por KLOC en prueba unitaria.

* + 1. Razón de revisión

Las métricas anteriores nos dicen lo que hicimos, la razón de revisión nos muestra lo que estamos haciendo. Se define como

|  |  |  |
| --- | --- | --- |
|  |  | (4.7) |

El tamaño del producto puede ser las líneas de código de un módulo del sistema o el número de páginas del documento de diseño detallado. La razón se normaliza por hora, es decir, si se invirtieron 30 minutos y se revisaron 10 páginas, la razón de revisión es 20 páginas por hora.

Normalmente razones de revisiones altas se relacionan con rendimientos bajos y viceversa. Lo importante es mejorar el proceso de manera que se pueda alcanzar la razón de revisión más alta posible manteniendo rendimientos de filtros iguales o mayores a 70% [[26](#WSH09)].

* + 1. Razón de fase

La razón de fase nos indica que tanto tiempo estamos invirtiendo en las revisiones respecto tiempo de desarrollo para esas mismas fases. En la metodología PSP existen 3 razones de fase: la primera es la razón de fase para diseño, la segunda la razón de fase para código y la tercera la razón de fase entre diseño y código. La última razón significa el tiempo que se invierte en el diseño del sistema respecto al tiempo que toma su codificación. En el caso de las tres revisiones, mientras mayor sea la razón mayor serán los rendimientos, es decir, mayor número de defectos encontrados.

Al igual que la razón de revisión, en estas métricas se espera que se encuentre una razón apropiada después de varias iteraciones a lo largo de diferentes proyectos. De acuerdo a los datos obtenidos por la metodología PSP [[26](#WSH09)], una razón de fase igual o mayor a 0.5 es obtendrá buenos resultados, para el caso de las primeras dos. Para el caso de la tercera, una razón mayor o igual a 1 provocará resultados aceptables en el proceso. Esto significa invertir, como mínimo, el mismo tiempo en diseñar el sistema que en codificarlo. **Si** no se invierte este tiempo el desarrollador estará diseñando al mismo tiempo que codifica, y siendo la codificación una actividad suficientemente retadora, la probabilidad de introducir defectos aumentará si se realiza de esta manera.

* + 1. Índice de calidad de proceso

El simple hecho de seguir las recomendaciones e invertir el tiempo señalado en las actividades de desarrollo y calidad no asegura que el producto final será de alta calidad. Es importante seguir buenas prácticas de diseño y de codificación, así como explotar las ventajas de las revisiones (que se explicarán a detalle en el siguiente capítulo). El índice de la calidad de proceso es un indicador sobre las actividades realizadas a lo largo del proyecto. Para el cálculo de este índice se necesitan dos mediciones claves: la primera es el número de defectos de compilación por cada mil líneas de código y la segunda el número de defectos de pruebas unitarias por cada mil líneas de código. Este índice se compone de la multiplicación de 5 factores [[26](#WSH09)]:

|  |  |  |
| --- | --- | --- |
|  |  | (4.8) |

|  |  |  |
| --- | --- | --- |
|  |  | (4.9) |

|  |  |  |
| --- | --- | --- |
|  |  | (4.10) |

|  |  |  |
| --- | --- | --- |
|  |  | (4.11) |

|  |  |  |
| --- | --- | --- |
|  |  | (4.12) |

Tomando como recomendación los tiempos y el número de defectos de PSP, cada factor tendría un máximo de 1, por lo que la meta sería que el valor final del índice también fuera de 1. La figura 4.3 [[26](#WSH09)] muestra el número de defectos por KLOC posteriores al desarrollo en relación con el valor del índice. Se puede observar que a medida que el índice aumenta el número de defectos que permanecen después del proceso disminuye.

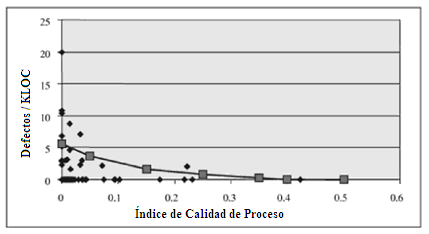


Figura 4.3. Índice de proceso de calidad.

Las métricas respecto a la calidad en el proceso de desarrollo sirven para determinar si lo que se está haciendo es adecuado o no. Sin embargo, todas surgen del tiempo que se invierte en las actividades y del número de defectos que se encuentran. Esto sugiere una actitud meramente reactiva. Lo ideal sería no inyectar los defectos en primer lugar, es decir, prevenirlos. **En la vida real, el uso de estas** actividades y métricas aunadas a actividades que ayuden a prevenir la inyección de defectos lograrán aumentar la calidad del producto final en gran medida.

Para introducir actividades que prevengan la inyección de defectos es necesario conocer los tipos de defectos que se están inyectando. Para conocer los tipos de defectos que están inyectando es necesario realizar actividades que nos permitan detectarlos y posteriormente clasificarlos. Las técnicas de detección de defectos nos permitirán conocer cuáles defectos se introducen con más frecuencia y cuáles son los que requieren más esfuerzo para su corrección.

CAPÍTULO 5

1. Técnicas de Detección de Defectos

La gran mayoría de las personas que alguna vez han estado involucradas en el proceso de desarrollo de software, se han visto en la situación de estar bajo presión por el calendario de entrega del proyecto. El primer pensamiento y la primera reacción es acelerar el proceso de codificación para terminar más rápido y dejar más tiempo para la fase de pruebas. Sin embargo, esta decisión (también la gran mayoría de las veces) tiene como consecuencia que el producto final no sea de la calidad necesaria y en última instancia provoque la desconfianza del cliente en el sistema.

Evidentemente, este no es un escenario deseado para cualquier compañía dedicada al desarrollo de software, ya que puede traer repercusiones para futuros proyectos (como su cancelación). Es principalmente por este escenario el que la mayoría de la gente involucrada en desarrollo de software, tanto a nivel operativo como gerencial, se pregunta si es posible realizar el desarrollo igual o más rápido y con una mayor calidad.

Las técnicas, formales y no formales, de detección de defectos que se encuentran en la literatura son herramientas valiosas que lograrán aumentar de manera significativa la calidad total del producto final. La primera técnica de detección concebida como tal de la que se tenga conocimiento surgió en 1976 gracias a Michael Fagan, por lo que llevan prácticamente 35 años en el mundo del software. Ahora bien, lo importante es saber qué son, cómo se implementan, quién las implementa, cuándo se implementan y lo más importante, porqué es bueno implementarlas.

Las técnicas de detección de defectos se pueden definir como la evaluación y revisión de un producto de trabajo por parte de uno o más compañeros calificados para tal actividad [[27](#KOw97)]. Esta definición nos da una idea de quiénes realizan las revisiones. Es importante escoger adecuadamente el número de personas involucradas para tal actividad y el perfil de las mismas. Pueden ser desde una hasta la cantidad que los recursos y el tiempo permitan, aunque más adelante se verá que solamente el agregar más y más personas tampoco es lo mejor necesariamente. También es importante

señalar que aunque la gerencia no participa directamente en las revisiones, es muy importante su apoyo activo para la realización de estas actividades.

Una de las grandes ventajas de implementar técnicas de detección de defectos es que no solamente se limitan a la revisión de código, sino que se pueden aplicar para revisiones de documentos **de especificación y de diseño, entre otros**. Por lo que se pueden aplicar prácticamente en todo el ciclo de vida de desarrollo. Sabemos que cada compañía de desarrollo (de cualquier tamaño) tiene sus particularidades en cuanto a la adopción de diferentes procesos, y las técnicas de detección de defectos no son la excepción. Aunque estas técnicas están bien definidas en la literatura, siempre es importante preguntarse si la implementación al pie de la letra es lo más adecuado o si es mejor adaptarlas a las características de la empresa. La mayoría de las veces la respuesta más acertada es lo segundo.

En este **punto se puede preguntar porque el implementar** este tipo de técnicas o procesos traerá ventajas al proceso mismo de desarrollo y aumentará la calidad de los productos. **Las ventajas importantes son las siguientes:**

* Son efectivas en cuanto a costos se refiere. Entre más tarde en el ciclo se encuentre un defecto, más costará su corrección. Por lo que al aplicar estas técnicas en etapas tempranas (requerimientos, diseño, codificación) el costo disminuye considerablemente, inclusive hasta 100 veces menos [[28](#BWB81)]. Además, el no detectar defectos en etapas tempranas puede causar que en etapas posteriores se originen más defectos derivados de uno solo. Por ejemplo, un defecto que se escapa en diseño, puede provocar 5 o más defectos en codificación y pruebas.
* Se pueden aplicar en etapas tempranas del ciclo. Como se ha comentado anteriormente, no solo se puede revisar código, sino cualquier producto de trabajo generado del proceso mismo de desarrollo.
* Bien implementadas, pueden llegar a ser mucho más eficientes que las pruebas mismas. Es decir, las técnicas puede detectar más defectos por unidad de esfuerzo que las diferentes etapas de pruebas.
* Son medibles. El esfuerzo requerido para generar métricas e información valiosa de la implementación de las técnicas puede llegar a ser despreciable. Estas mediciones son de gran importancia, ya que permiten conocer qué tan eficientes son las técnicas, cuándo es importante realizar cambios en las mismas y cuáles son más eficientes para determinadas situaciones.

Es importante considerar algunos factores para la implementación de técnicas de detección, que pueden maximizar la efectividad de las mismas. Estos factores son [[29](#Lai99)]:

* Mientras mayor sea el esfuerzo de revisión, mayor será el número de defectos detectados. Es una aseveración que a simple vista tiene sentido y que se puede lograr de dos maneras. La primera es aumentando el número de revisores y la segunda es aumentando el esfuerzo individual de cada revisor. La probabilidad de encontrar un defecto se define como [[29](#Lai99)]:

|  |  |  |
| --- | --- | --- |
|  |  | (5.1) |

* Donde k es el número de revisores y *p* la probabilidad de encontrar un defecto, que para este caso, se asume que es la misma probabilidad para todos los revisores. Es importante considerar que existe un efecto techo en el que agregar más revisores no aumentará el número de defectos detectados. La figura 5.1 [[29](#Lai99)] muestra la gráfica de la probabilidad de encontrar un defecto suponiendo que .
* Mientras más grande sea el producto de trabajo revisado, mayor será el número de defectos detectados. Se espera que los productos más grandes sean más complejos y tengan más defectos, pero se observa el mismo efecto que en el punto 1. Esto también debido al tiempo de revisión, ya que la efectividad del ser humano se ve afectada negativamente después de un tiempo de realizar una misma actividad que requiera concentración.
* Entre mayor sea el producto de trabajo realizado, mayor será el esfuerzo de revisión. El comportamiento de este punto es idéntico al observado en el punto 2. Ya que la principal restricción es la efectividad de revisión a medida que el tiempo pasa.

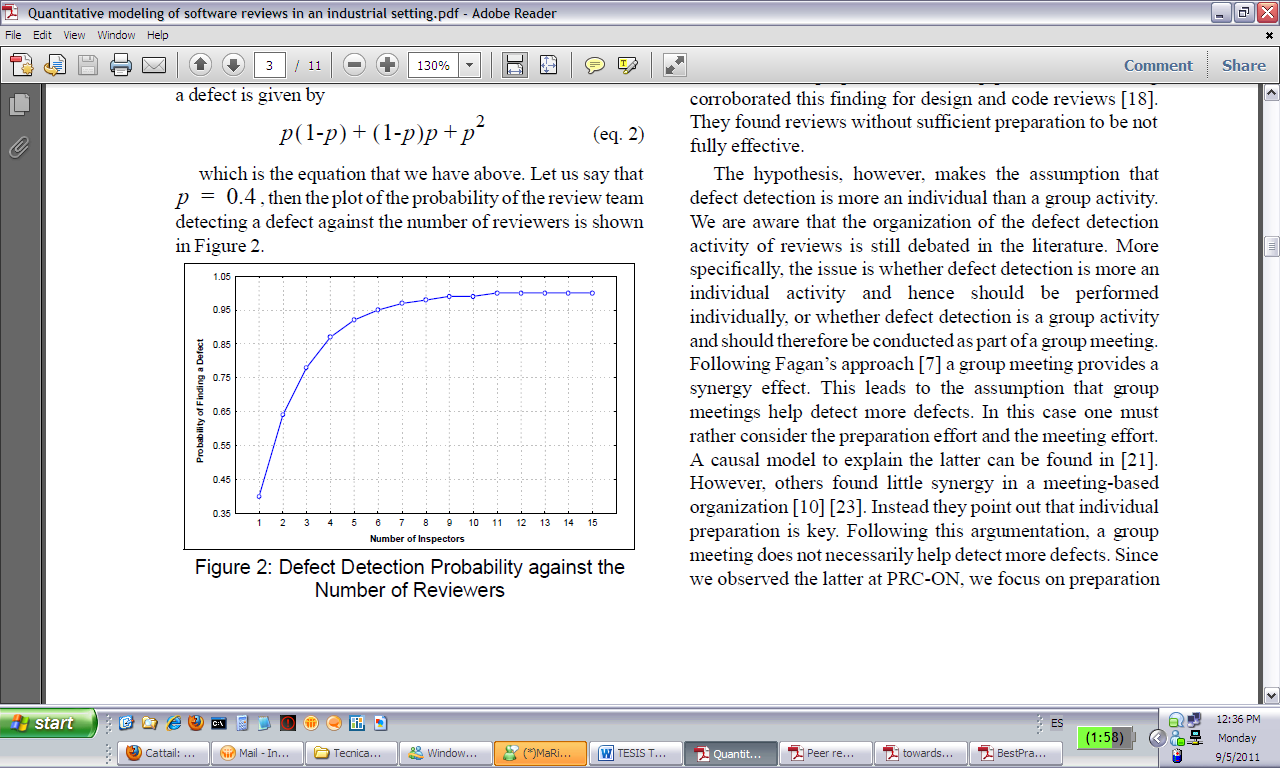


Figura 5.1. Probabilidad de detectar defectos respecto al número de revisores.

* 1. Tipos de técnicas

Existen diferentes tipos de técnicas de detección de defectos, unas más formales y rígidas que otras. Evidentemente la diferencia radica en la forma de conducir la revisión del producto de trabajo. También existen diferentes clasificaciones acerca de las técnicas; una clasificación coherente consiste en separar las técnicas que involucran a una sola persona y las técnicas en las que participan dos o más miembros del equipo.

* + 1. Revisión Personal

La técnica más importante de detección de defectos es la revisión personal [[26](#WSH09)]. PSP fue desarrollado como tal por Watts Humphrey. En este proceso se hace un énfasis muy marcado en la evaluación, seguimiento y mejora de la calidad personal como elemento vital de la calidad total del producto de software. La revisión personal consiste en examinar el producto de trabajo antes de entregarlo a cualquier otro miembro del equipo, ya sea para su lectura, compilación, revisión, implementación o prueba.

Humphrey realza la necesidad de realizar las revisiones personales de cualquier producto con una metodología definida y bajo ciertas condiciones, por lo que se pueden considerar como principios de la revisión personal los siguientes:

* Revisar personalmente todo el trabajo propio antes de pasar a la siguiente fase de desarrollo.
* Intentar lo mejor posible corregir todos los defectos antes de dar el producto de trabajo a otra persona en el equipo de desarrollo.
* Utilizar una lista de chequeo personal y seguir un proceso estructurado de revisión.
* Seguir las buenas prácticas de la revisión: revisar en incrementos pequeños, hacer las revisiones en papel y hacerlas cuando estás descansado.
* Medir el tiempo de la revisión, el tamaño de los productos revisados y el número y tipo de defectos encontrados y perdidos.
* Usar los datos de las mediciones para mejorar el proceso personal de revisión.
* Diseñar e implementar los productos para que sean fáciles de revisar.
* Revisar los datos para identificar las formas de prevenir defectos.
  + 1. Revisión entre colegas

Existen varias técnicas de detección de defectos que se realizan entre colegas y/o miembros de un equipo de desarrollo. Estas técnicas se pueden clasificar de acuerdo a su formalidad. Existen 6 diferentes tipos de técnicas: revisión ad hoc, revisión general, revisión de parejas, caminata, revisión de equipo e inspección [[3](#Har05)]. La figura 5.2 [[3](#Har05)] muestra los tipos de revisión entre colegas de acuerdo a su formalidad.

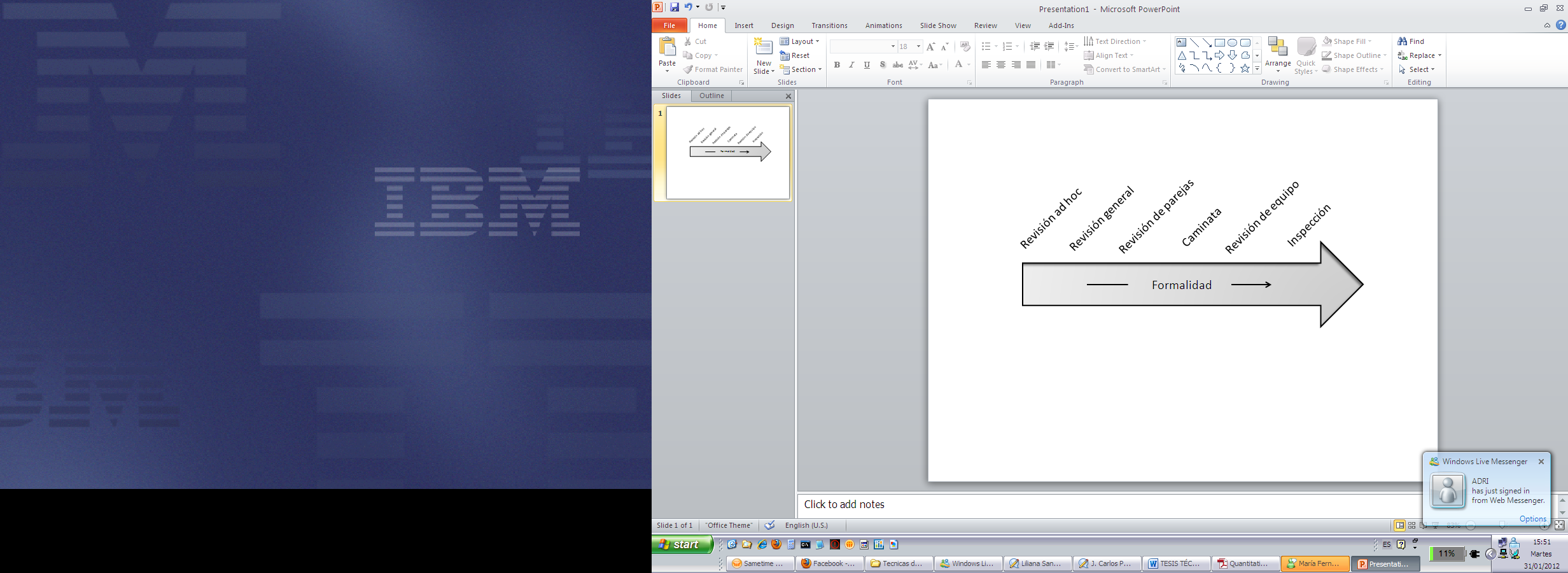


Figura 5.2. Tipos de revisión entre colegas.

**La caminata y la inspección son dos de las técnicas más formales**, por lo que se detallarán a continuación.

* + 1. Caminata

La caminata es un tipo de revisión en el que precisamente una persona (preparada especialmente para ello) revisa el producto, exponiéndolo a una audiencia. Mediante esta técnica se pueden obviar muchos detalles, con lo cual se reduce el tiempo de revisión [[30](#GMW84)]. Sin embargo, esto puede ser contraproducente si el objetivo es precisamente detectar defectos que residen en los detalles. Al mismo tiempo, el proceso de revisión está definido por el producto de trabajo siendo revisado, a diferencia de la inspección, en el que el proceso se determina por los puntos a revisar.

Una de las ventajas de la implementación de una caminata es que el presentador se prepare explícitamente para realizar la revisión, lo que acelera el proceso. También, al tiempo que no requiere de una preparación para nada exhaustiva de los participantes por adelantado, la audiencia de la presentación se puede familiarizar rápidamente con el producto. Si la audiencia es lo suficientemente diversa, se puede afirmar con alto grado de certeza que la revisión será altamente efectiva [[30](#GMW84)].

Al mismo tiempo que se presentan estas ventajas, precisamente ciertas desventajas emanan de ellas. Siendo la principal desventaja el que al no requerirse preparación previa, el grado de entendimiento adquirido por cada participante al momento de la presentación no sea el mismo, causando la falta de profundidad deseada en la revisión.

* + 1. Inspección

Es probablemente la técnica más formal de detección de defectos de software [[3](#Har05)]. Consiste de los siguientes pasos [[27](#KOw97)]:

* Planeación: Seleccionar dónde, cuándo y quiénes participarán.
* Resumen: Revisión general para que los revisores se familiaricen con el producto de trabajo. **Es opcional.**
* Preparación: Cada revisor lee el producto de trabajo e identifica posibles defectos. Todo esto generalmente mediante listas de chequeo. La persona que realiza el papel de líder, se prepara para la junta de revisión.
* Junta: El líder modera la junta y realiza la revisión del producto. Los revisores lo interrumpen para discutir los defectos encontrados. Lo más importante es que NO se permite discutir posibles soluciones para ningún defecto. **También se determina la calidad del producto y se determina el tipo de seguimiento.**
* Re-trabajo: El autor del producto realiza las correcciones pertinentes.
* Seguimiento: El autor del producto notifica al líder de las correcciones y éstas son revisadas.

La inspección no es una técnica que se pueda limitar solamente a la revisión de código o a una etapa dentro del ciclo de desarrollo, sino que es perfectamente aplicable a los productos de trabajo generados desde el principio y hasta el final del ciclo. **Por ejemplo, los siguientes 5 tipos de inspecciones** [[5](#GCh99)]:

* Inspección de arquitectura: **Revisa documentos** de requerimientos y especificación.
* Inspección de diseño: **Revisa los documentos** de diseño. Es la que más reduce el costo de detección de defectos (44%), inclusive por encima de la inspección de código (con un 39%) [[31](#alW02)].
* Inspección de código: **Revisión de código** que aún no ha sido ejecutado.
* Inspección de casos de prueba: **Revisión de los casos** de prueba diseñados.
* Inspección de implementación de casos de prueba: **Revisión de los casos** de prueba ya implementados o ejecutados.

La implementación de esta técnica de revisión no es algo que se deba tomar a la ligera. Primero que nada es importante decidir de manera adecuada los revisores que estarán involucrados. Si los revisores no son los mejores posibles, puede ser que la efectividad de la técnica se vea reducida considerablemente. De igual manera, al saber que esta técnica es la más formal y rigurosa, es muy probable que sea también la que más tiempo consuma. Esto nos obliga a decidir sobre cuáles productos de trabajo se realizará una inspección. Es prácticamente imposible realizar inspecciones a todos los productos de trabajo, y es necesario escoger los más críticos para el proyecto.

* 1. Implementación de técnicas de detección de defectos

La implementación de estas técnicas no representa la panacea sobre el aseguramiento de la calidad en productos de software. Como cualquier otra actividad y/o proceso, estas técnicas llevan consigo ventajas y desventajas en su aplicación. Lo importante es implementarlas de cierta manera que sus beneficios superen en gran medida a sus desventajas. Es por esta razón que su implementación no consiste simplemente de seguir al pie de la letra los pasos para llevarlas a cabo, sino que se requiere evaluar de manera adecuada la forma en que se pretenden implementar. Los principales beneficios reconocidos en la literatura al implementar estas técnicas son [[3](#Har05)]:

* Educación e intercambio de conocimiento.
* Mayor conciencia y seguimiento al proyecto
* Mejora en el proceso de desarrollo
* Aumento en la detección de defectos
* Detección de defectos de manera más temprana y rápida.
* Reducción en el costo del proyecto.

De igual manera, los principales obstáculos por los que muchas empresas no implementan revisiones o técnicas de detección de defectos son:

* Falta de tiempo.
* Falta de recursos humanos.
* Costo.
* Complejidad y falta de capacitación.
* Resistencia al cambio.
* Ineficiencia.

En un estudio realizado en el que se consideraron 10 empresas de distintos tamaños (desde 5 empleados hasta varios miles), y 12 equipos de desarrollo, se realizaron entrevistas estructuradas acerca de las técnicas de detección que implementaban y cuáles identificaban como los principales motivadores y desmotivadores para llevarlas a cabo. La entrevista consistía en conocer las técnicas con las que estaban familiarizados, los productos de trabajo que se revisaban, quiénes participaban en las revisiones y el tipo de información que se recababa como resultado de la revisión. La tabla 5.1 [[3](#Har05)] muestra la distribución sobre la importancia que le dan los 12 equipos de desarrollo entrevistados a cada motivador. **Los números representan la cantidad de equipos de desarrollo, por lo que cada fila suma 12.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Motivador** | **Muy importante** | **Importante** | **Relativa importancia** | **Poco importante** | **No importa** |
| Intercambio de conocimiento | 1 | 7 | 1 | 2 | 1 |
| Intercambio de información | 2 | 4 | 5 | 1 | 0 |
| Educación | 4 | 3 | 2 | 2 | 1 |
| Gestión y seguimiento de proyecto | 0 | 4 | 5 | 2 | 1 |
| Mejora de proceso | 3 | 2 | 3 | 4 | 0 |
| Encontrar más defectos | **8** | 3 | 0 | 1 | 0 |
| Encontrar defectos previo a las siguientes fases de desarrollo | **9** | 2 | 0 | 1 | 0 |
| Encontrar defectos tempranamente | **7** | 3 | 0 | 1 | 1 |

Tabla 5.1. Motivadores de las técnicas de detección de defectos.

Se observa una marcada tendencia sobre el hecho de detectar más defectos y hacerlo de manera temprana, lo que coincide con una reducción en los costos totales y una mayor calidad del producto de software. En cambio, la tabla 5.2 [[3](#Har05)] muestra los principales desmotivadores para incorporar diferentes técnicas como parte de su proceso de desarrollo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Obstáculo** | **Muy importante** | **Importante** | **Relativa importancia** | **Poco importante** | **No importa** |
| Falta de tiempo | 8 | 4 | 0 | 0 | 0 |
| Falta de recursos | 4 | 5 | 1 | 1 | 1 |
| No requerido por la gerencia | 0 | 2 | 2 | 1 | 7 |
| Solo el autor entiende el producto | 0 | 2 | 1 | 1 | 8 |
| Costo | 0 | 0 | 1 | 3 | 8 |
| Falta de conocimiento de las revisiones | 0 | 1 | 3 | 0 | 8 |
| Laboriosidad | 0 | 3 | 2 | 3 | 4 |
| Equipos distribuidos geográficamente | 1 | 0 | 0 | 1 | 10 |
| No aplican en la organización | 0 | 0 | 0 | 0 | 12 |
| No hay necesidad de hacer revisiones | 0 | 0 | 1 | 1 | 10 |

Tabla 5.2. Desmotivadores de las técnicas de detección de defectos.

La falta de tiempo y recursos se identifican como los mayores obstáculos para realizar este tipo de actividades. Estos dos obstáculos representan el mayor reto para las empresas, algo que no es nada fácil de resolver. Sólo un equipo consideró como un gran obstáculo el hecho de estar distribuidos geográficamente, mientras que 3 equipos consideraron que las revisiones son muy laboriosas.

A su vez, **existen otro tipo de situaciones** que no favorecen la ejecución de revisiones entre colegas. Estas situaciones abarcan la aceptación emocional, problemas de ejecución, tipo de proyecto, calendario, recursos humanos, pequeñas empresas y la organización actual de las mismas [[5](#GCh99)]. Sin embargo, no deben ser situaciones que impidan de manera tajante la adopción de estas técnicas. Siempre es importante reflexionar sobre la manera en la que se realicen las revisiones para incrementar gradualmente su efectividad, y lo más importante, evitar desperdiciar el tiempo de los miembros del equipo. Para el caso de las revisiones de código, las siguientes prácticas pueden ayudar a evitar una caída en la productividad y a evitar que los miembros se sientan atacados o cuestionados sobre su trabajo, al mismo tiempo que se incrementa la calidad del código [[32](#Sma)]:

* Revisar entre 200 y 400 líneas de código en una sola vez. Revisar más líneas puede provocar que la habilidad para encontrar defectos disminuya.
* La razón de revisión debe oscilar entre las 300 y 500 líneas por hora. Es importante prestarle atención a lo que se revisa, y no solo tratar de cumplir con el proceso. **Inclusive PSP recomienda que la revisión sea de 200 líneas por hora** [[26](#WSH09)]**.**
* Que el tiempo de revisión sea de 60 a 90 minutos. Después de este tiempo la capacidad de concentración disminuye considerablemente en el ser humano.
* Toda revisión entre colegas debe estar precedida por una revisión personal. Es importante para evitar que la revisión se centre en detalles no tan importantes.
* Determinar objetivos para la revisión y capturar métricas para mejorar el proceso de revisión.
* Utilizar una lista de chequeo al momento de revisar. Las listas permiten seguir un orden, por lo que colaboran en la efectividad.
* Verificar que los defectos sean corregidos. No dar por terminado el ciclo al detectar los defectos, sino darle seguimiento hasta que sean corregidos.
* Provocar un ambiente en el que encontrar defectos sea visto de manera positiva. Es una tarea principalmente de los gerentes, ya que si se ve de manera contraria puede afectar la efectividad. El propósito es eliminar defectos, sin importar quién los haya inyectado.
* Evitar que los miembros del equipo, en especial los desarrolladores, se sientan vigilados por las métricas resultantes de las revisiones.
* Siempre tratar de revisar lo que se codifica, por más pequeña que sea la revisión (no el código).

Con estas prácticas se pueden adaptar las revisiones a las características particulares de cada empresa, aumentando la probabilidad de éxito en su implementación.

Las técnicas de detección de defectos especifican el cómo, en relación a reducir el número de defectos, y las métricas de proceso (especificadas en el capítulo anterior) determinan qué tan bien está el cómo.

* 1. Resultados de las técnicas de detección de defectos

Hasta el momento se han revisado los motivadores y desmotivadores, las ventajas y desventajas y un conjunto de buenas prácticas que ayudan a mejorar la implementación de revisiones, junto con la descripción de algunas de ellas. Pero ¿Cuál puede ser el resultado de incorporarlas al proceso de desarrollo? ¿Qué tan efectivas son? ¿Cuántos defectos encuentran? Estas son preguntas que cualquiera se puede preguntar al momento de evaluar la incorporación de las técnicas a su proceso.

Lo más importante es tener en cuenta que los estudios que se han realizado al respecto, han sido sobre diferentes proyectos, diferentes empresas, diferentes técnicas y con diferentes metodologías, por lo que los resultados varían de un estudio a otro. El punto clave es evaluar las circunstancias en las que se realizó el estudio para comprender los resultados.

Por ejemplo, un estudio sobre la efectividad de las revisiones se basó en la recolección de 21 conjuntos de datos sobre revisiones a requerimientos más 10 conjuntos de datos sobre revisiones a código [[31](#alW02)]. Todos los datos se recabaron de la literatura, de diferentes empresas y algunas universidades. El propósito fue establecer si hay diferencias en la efectividad entre las revisiones a requerimientos y a código y si hay diferencia en efectividad entre diferentes técnicas de detección. Pero el más interesante fue determinar el número de revisores necesarios para obtener una determinada efectividad en la revisión con cierta probabilidad de ocurrencia. Los resultados fueron los siguientes:

* No hay diferencia práctica en efectividad entre revisiones de requerimiento y de código. Por lo que no se puede afirmar que es mejor enfocar el esfuerzo a una de las dos. La gráfica izquierda de la figura 5.3 muestra los resultados [[31](#alW02)], siendo los cuartiles de la izquierda los referentes a documentos de requerimientos. **Sin embargo la fase de código inyecta más defectos.**
* Entre las técnicas de revisión ad hoc, la caminata y la de revisión de equipo con listas de chequeo hay una diferencia marcada, siendo la caminata la más efectiva. La gráfica derecha de la figura 5.3 muestra los resultados. **Las técnicas de izquierda a derecha son: revisión ad hoc, caminata y revisión de equipo. Hay que notar que no hay diferencia con distinto número de revisores, la caminata es la más efectiva en todos los casos.**
* La tabla 5.3 [[31](#alW02)] muestra las probabilidades de obtener una determinada efectividad de acuerdo al número de revisores. Las columnas representan los revisores, las filas la efectividad y los datos de la tabla la probabilidad. Las probabilidades son para revisiones en general y no especifican ningún tipo de revisión. La tabla debe ser usada solamente como guía para determinar el número de revisores.

Un segundo estudio consiste en la recopilación de 340 revisiones (de las cuales 145 fueron a requerimientos, 94 a diseño y 101 a código) de una empresa alemana. Estos datos fueron extraídos de un sistema en el que se alimentaba de manera manual la información. Los datos pasaron por un proceso de validación para eliminar datos inconsistentes [[29](#Lai99)]. Se obtuvieron gráficas con el número de defectos de las revisiones por etapa, el esfuerzo requerido y el tamaño.

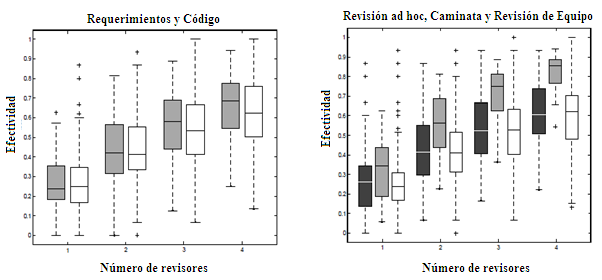


Figura 5.3. Efectividad de las revisiones.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0.1** | 0.87 | 0.97 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| **0.2** | 0.64 | 0.90 | 0.96 | 0.99 | 1.00 | 1.00 | 1.00 |
| **0.3** | 0.37 | 0.75 | 0.89 | 0.93 | 0.97 | 1.00 | 1.00 |
| **0.4** | 0.17 | 0.57 | 0.77 | 0.84 | 0.87 | 0.91 | 1.00 |
| **0.5** | 0.10 | 0.35 | 0.57 | 0.74 | 0.81 | 0.86 | 0.95 |
| **0.6** | 0.06 | 0.22 | 0.41 | 0.57 | 0.65 | 0.77 | 0.71 |
| **0.7** | 0.03 | 0.10 | 0.23 | 0.39 | 0.43 | 0.55 | 0.50 |
| **0.8** | 0.03 | 0.06 | 0.15 | 0.24 | 0.22 | 0.27 | 0.27 |
| **0.9** | 0.01 | 0.02 | 0.04 | 0.08 | 0.11 | 0.15 | 0.05 |
| **1.0** | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.04 | 0.00 |

Tabla 5.3. Número de revisores contra efectividad.

La tabla 5.4 muestra los resultados obtenidos por este estudio. Los resultados fueron divididos en cuartiles y además se obtuvo la media. El esfuerzo requerido incluye el de preparación y el total (en horas). El tamaño se mide en páginas para los documentos de requerimientos y diseño y en líneas de código para el código. También se calculó la densidad de defectos de los productos revisados.

Un tercer estudio, que en realidad es una recopilación de estudios y experimentos de la literatura, compara la efectividad y la eficiencia de las técnicas de detección de defectos contra las fases de pruebas (en el estudio a las técnicas se les llaman inspección). La eficiencia consiste en el número de defectos detectados por unidad de tiempo y la efectividad el porcentaje de defectos encontrados por la revisión del total de defectos encontrados [[33](#Run06)]. Son 9 estudios sobre defectos en código, 1 estudio sobre defectos en diseño y 2 sobre todo el proceso de detección.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Medición** | **Etapa** | **Máximo** | **Mínimo** | **Media** | **Cuartil 25-75** |
| **Defectos encontrados** | Requerimientos | 58 | 0 | 12 | 5-25 |
| Diseño | 87 | 0 | 15 | 6-26 |
| Código | 135 | 0 | 14 | 8-24 |
| **Esfuerzo preparación** | Requerimientos | 15 | 1 | 3 | 2-5 |
| Diseño | 40 | 2 | 2.5 | 2-5.5 |
| Código | 45 | 2 | 5 | 2.5-7.5 |
| **Esfuerzo total** | Requerimientos | 32 | 2.5 | 7 | 3.5-11 |
| Diseño | 42 | 2.5 | 6 | 3.5-10 |
| Código | 59 | 2.5 | 8 | 4-12.5 |
| **Tamaño** | Requerimientos | 240 | 1 | 24 | 18-40 |
| Diseño | 278 | 0 | 22.5 | 10-44 |
| Código | 10000 | 0 | 1450 | 800-3420 |
| **Densidad de defectos** | Requerimientos | 11 | 0 | 0.25 | 0.25-1 |
| Diseño | 29 | 0 | 0.5 | 0.5-1.25 |
| Código | 46 | 0 | 9 | 4-16 |

Tabla 5.4. Concentrado resultados de revisiones.

En el estudio se clasifican los defectos por su origen, ya sea en requerimientos, diseño o codificación. La tabla 5.5 muestra un resumen de los estudios comparándolos en su eficiencia y efectividad, la suma de los estudios no necesariamente es 12 ya que algunos solo comparan una característica. Las figuras 5.4 y 5.5 [[33](#Run06)] muestran la eficiencia y efectividad promedio de las técnicas y las pruebas de la mayoría de los 12 estudios para el caso de detección de defectos en código.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Eficiencia** | | | **Efectividad** | | |
|  | Inspección > Pruebas | Inspección = Pruebas | Inspección < Pruebas | Inspección > Pruebas | Inspección = Pruebas | Inspección < Pruebas |
| **Número de estudios** | 1 | 0 | 5 | 1 | 2 | 6 |

Tabla 5.5. Comparación entre revisiones y pruebas.

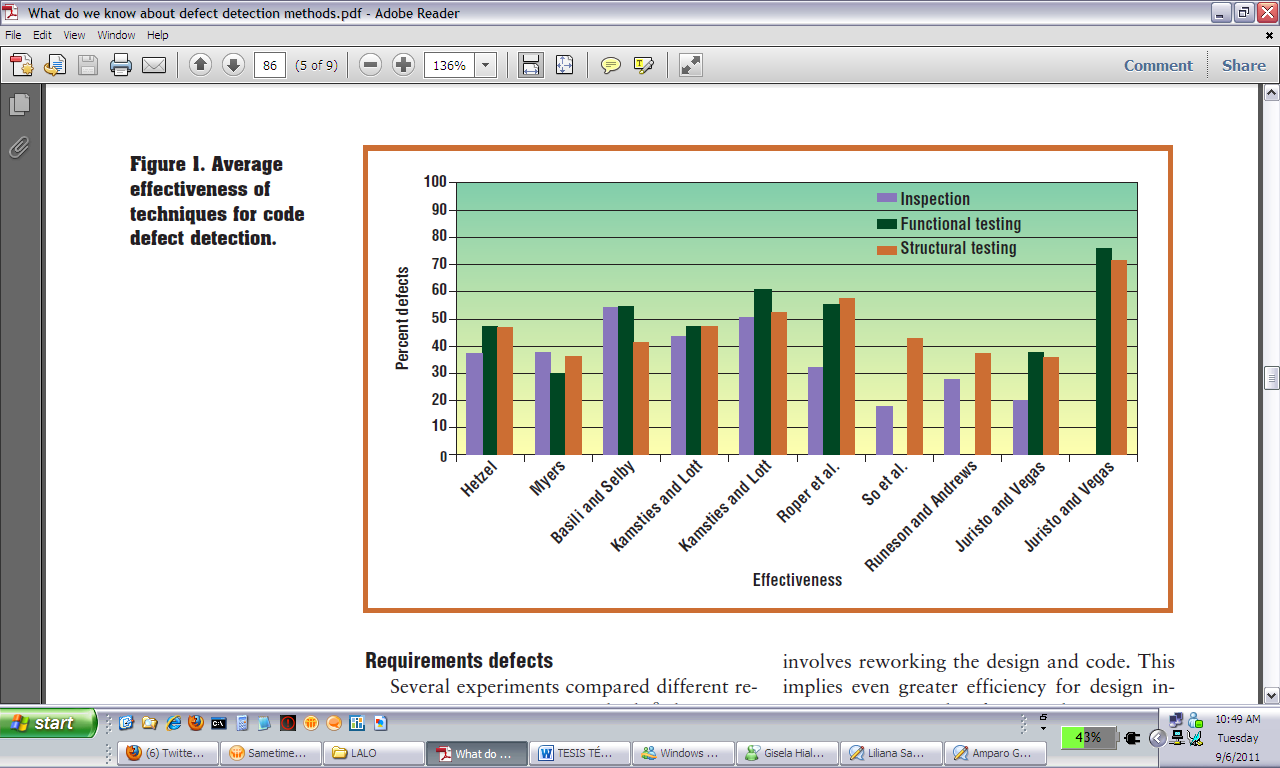


Figura 5.4. Efectividad promedio de revisiones y pruebas.

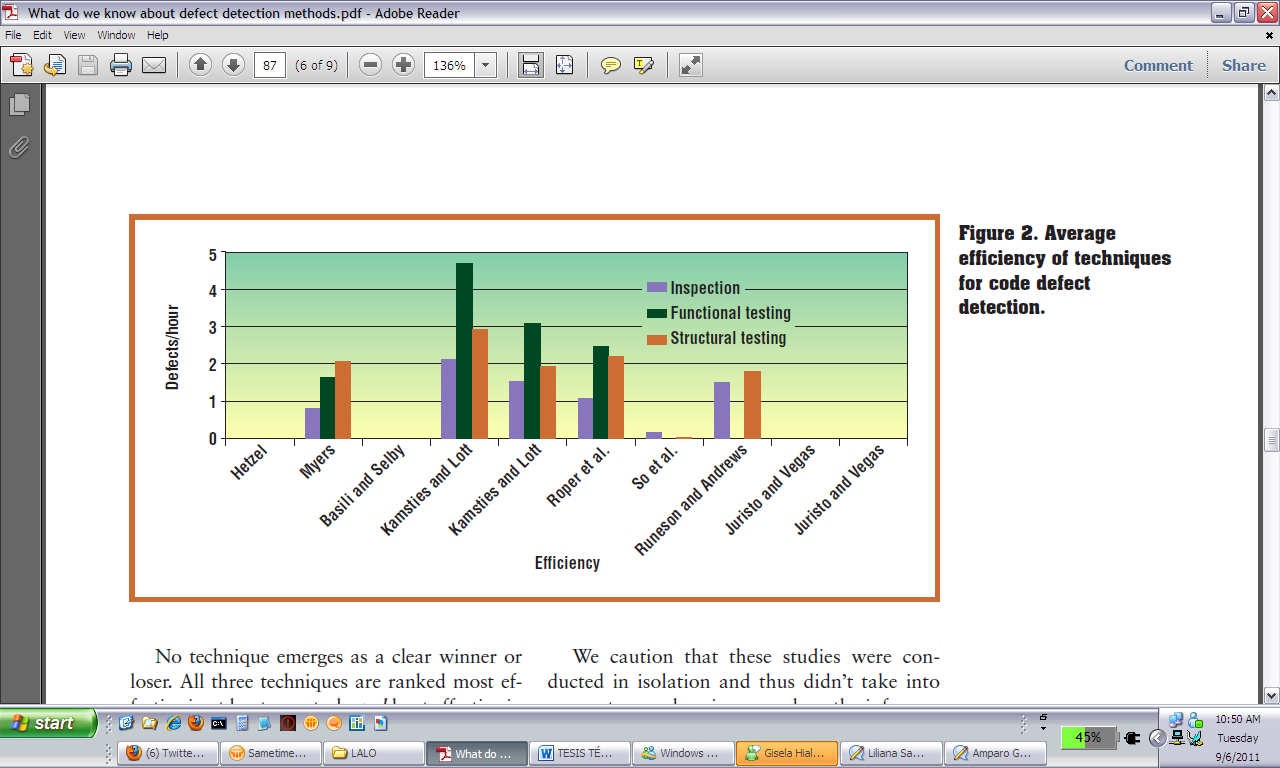


Figura 5.5. Eficiencia promedio de revisiones y pruebas.

Es evidente que los resultados muestran una gran variedad en cuanto a la implementación de técnicas de detección de defectos. Desde mi punto de vista, esta gran variedad se debe al gran número de factores envueltos en su implementación. Dentro de estos factores podemos nombrar el tipo y el tamaño del proyecto, la técnica específica utilizada, el número de revisores involucrados, el tamaño del producto a revisar, la razón de **revisión del producto y las listas de chequeo utilizadas, entre otros,** pudiendo nombrar probablemente 20 más.

Los estudios, experimentos, discusiones y debates sobre las técnicas de detección de defectos giran en torno a qué tan benéficas son y qué tanto aumentan la calidad del producto, y no en torno a si lo son o no. Por lo que es un hecho que la implementación de técnicas de detección de defectos trae beneficios a las empresas que las realizan. El punto clave radica en la evaluación y medición de las mismas para ir aumentando su eficiencia y efectividad de manera paulatina, algo que definitivamente no es fácil de realizar, pero que seguramente también permitirá aumentar la calidad del proceso y por ende del producto al mismo tiempo que se reducen los costos totales.

CAPÍTULO 6

1. Bug-Manager

El sistema **Bug Manager (BM)** se puede definir como un sistema de administración, seguimiento y de mejora continua de la calidad personal y de grupo en el desarrollo de sistemas de software; todo esto mediante el seguimiento de proyectos, actividades de desarrollo y de calidad a lo largo del ciclo de vida, de defectos inyectados, removidos y corregidos, y de la creación de estadísticas y métricas verdaderamente útiles a la empresa.

El sistema fue diseñado y pensado para que las empresas que no tuvieran ninguna actividad de calidad no les costara demasiado trabajo comenzar a realizarlas. Sabemos que es cada vez mayor la necesidad de las empresas de contar con sistemas que realmente les representen una mejora significativa en sus procesos actuales, es decir, con sistemas de alta calidad. Todo esto lleva a que las empresas dedicadas al desarrollo de software estén envueltas en un proceso de mejora continua respecto a los productos que lanzan al mercado, es decir, productos que cada vez sean de mayor calidad y por tanto contengan menos defectos.

**La descripción completa del sistema se encuentra en el apéndice 1 de este documento. En esta sección se mencionan las principales funcionalidades, los beneficios y su aportación como herramienta para desarrollo de software. La funcionalidad principal se lista a continuación:**

* **Registrar y dar seguimiento a las actividades de desarrollo y calidad establecidas para el ciclo de vida de desarrollo.**
* **Hacer un seguimiento puntual a la inyección, detección y remoción de defectos a lo largo de las diferentes etapas del ciclo de vida.**
* **Generar estadísticas y métricas de valor para la empresa y el personal en base a la información proporcionada por los usuarios del sistema.**
* **Servir como guía en los procedimientos principales de aseguramiento de la calidad.**

**Los beneficios que se obtienen al utilizar la herramienta son:**

* **Definición de las fases para un proyecto en específico, con la flexibilidad de definir diferentes fases para diferentes proyectos.**
* **Clasificación de las fases especificadas para el proyecto como fases de requerimientos, diseño, codificación, revisión, pruebas o mantenimiento. Esto permite obtener estadísticas sobre los costos de la calidad y su retorno de inversión.**
* **Con la posibilidad de definir tipos de defectos, la empresa podrá crear la nomenclatura que considere más conveniente para el registro de los mismos.**
* **Con la clasificación de las actividades del proyecto, se pueden obtener estadísticas y gráficas sobre las técnicas de detección implementadas para un proyecto en específico.**
* **Con la construcción de plantillas personalizadas, se podrán realizar las revisiones de calidad de una manera que se adapte a las características de la empresa.**
* **Se obtienen datos valiosos sobre un usuario, un proyecto o sobre la empresa de manera muy sencilla y rápida.**

**Con el sistema BM se podrá llevar un control sobre la evolución del proyecto, mediante el registro rápido y sencillo de lo que se hace para a final de cuentas conocer cómo se desarrolló. Este conocimiento permitirá a la empresa tomar decisiones que provoquen una mejora para el siguiente proyecto.**

1. 1. Metodología

El equipo de desarrollo del sistema contó con 4 miembros, de los cuales 3 colaboramos con la definición y desarrollo del mismo. El cuarto miembro (nuestro asesor de tesis) colaboró en la definición de las capacidades y funcionalidades del sistema:

* Dr. Oscar Adrián Mondragón Campos (asesor de tesis).
* Ing. Humberto García Robles (**ingeniero de software**).
* Ing. Marco Antonio Rangel Bocardo (**ingeniero de software**).
* Ing. Eduardo Campos Peiro (**ingeniero de software**).

Para la construcción del sistema se desarrollaron dos documentos principales: El documento de Concepto de Operaciones y el de Diseño de Base de Datos. El primero corresponde al apéndice 1 y el segundo al apéndice 2. En el Concepto de Operaciones se define lo siguiente:

* La situación y condiciones del sistema actual
* La justificación y naturaleza del sistema propuesto (BM).
* Las funcionalidades requeridas y opcionales para este sistema.
* Una descripción completa del sistema propuesto con antecedentes, alcances, objetivos, módulos, modos de operación, tipos de usuarios y ambiente.
* Los escenarios de operación del sistema para los diferentes modos de operación.
* Las pantallas del sistema.
* Análisis sobre el impacto, las capacidades y limitaciones del sistema propuesto.

El Diseño de Base de Datos presenta un esquema con las tablas, los campos de cada tabla, las relaciones entre las mismas y las llaves primarias de cada tabla. Posterior a la definición de estos documentos se comenzó con la definición de las herramientas y lenguajes a utilizar para el desarrollo del sistema.

Posteriormente se repartieron las actividades de desarrollo conforme a los módulos definidos para el sistema y se comenzó con la codificación. Para el desarrollo del proyecto se adoptaron algunas de las prácticas definidas en la metodología de “Personal Software Process” (PSP) [[26](#WSH09)].

Se midió el tamaño, el tiempo invertido en cada fase y el número de defectos encontrados en las revisiones personales, todo esto para cada actividad. De igual manera se llevó un registro de los defectos encontrados, en el que se especifica la fase de inyección y remoción, la actividad de inyección y remoción, el esfuerzo invertido en la remoción, el tamaño y si el defecto se deriva de uno previo o no.

Se realizaron revisiones personales para cada actividad y/o módulo de desarrollo. Se realizaron revisiones entre colegas para los principales módulos del sistema. Por último se realizaron pruebas de integración del mismo. En el capítulo 7 se detallan los resultados obtenidos en el desarrollo de este sistema.

* 1. Arquitectura

La arquitectura definida para el desarrollo del sistema es la de modelo-vista-controlador (MVC). Para el caso del BM, tanto la lógica de negocio como los datos están contenidos en el modelo. La figura 6.1 ejemplifica la arquitectura utilizada.

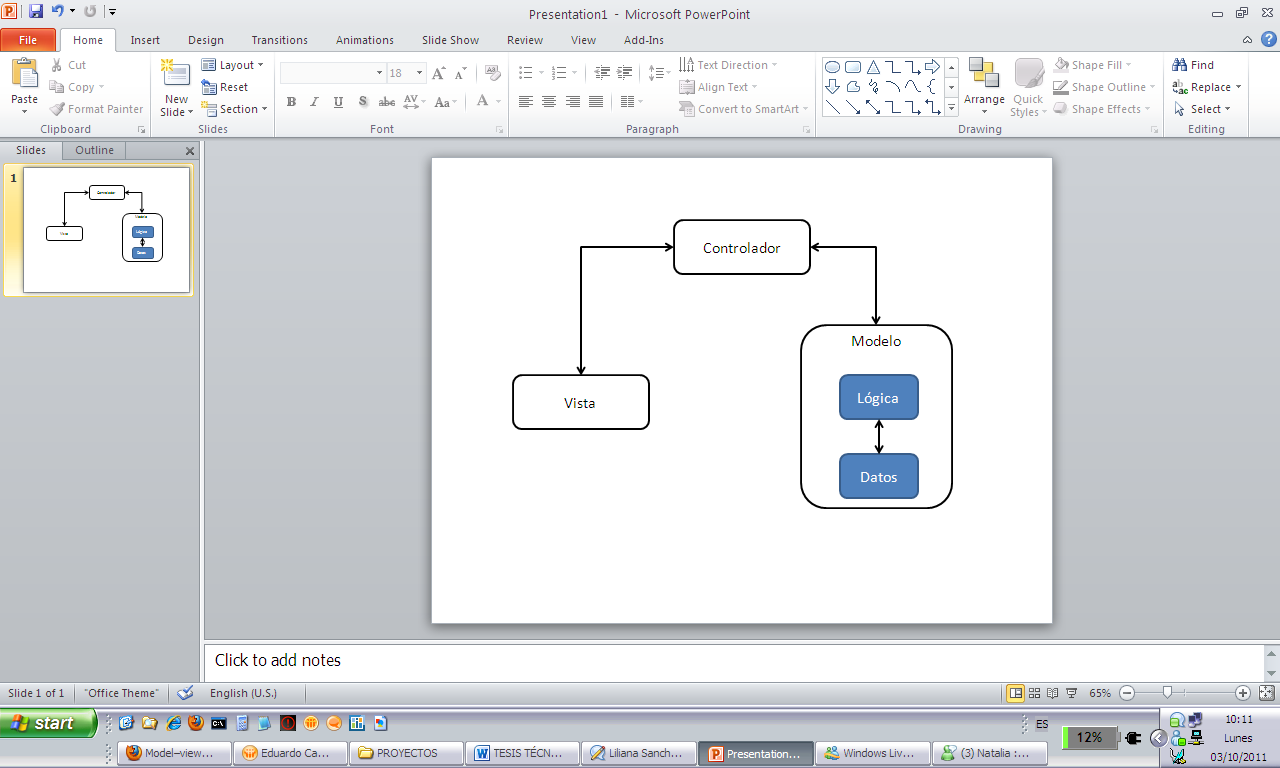


Figura 6.1. Arquitectura modelo-vista-controlador.

* 1. Características Técnicas

Las características técnicas del sistema se resumen en la tabla 6.1. Para el desarrollo del sistema se utilizaron tecnologías de código abierto tanto en el lenguaje de programación, en la base de datos, así como en el servidor web.

|  |  |
| --- | --- |
| **Características técnicas** | |
| Lenguajes de programación | Java, JavaScript, HTML |
| Versión JDK | 1.6 |
| Framework para arquitectura MVC | Spring 3.0.4 |
| Base de datos | MySQL 5.0.7 |
| Servidor Web | Apache Tomcat 6.0.29 |
| Ambiente de desarrollo integrado | NetBeans 6.9.1 |
| Cliente administración de la configuración | TortoiseSVN 1.6.16 |
| Repositorio | Google Code |
| Administración del proyecto | Maven 2.2.1 |

Tabla 6.1. Características técnicas del sistema BM.

Maven es una herramienta que permite administrar el proyecto (desde el punto de vista del código), así como la automatización en la compilación y construcción del proyecto en sí. Tiene la capacidad de descargar de manera automática las librerías utilizadas en el desarrollo. Utiliza el concepto de “Project Object Model” (POM) en el que se describe el proyecto, las dependencias con componentes y módulos externos, así como el orden en el que se construye el proyecto.

* 1. Módulos

Los módulos lógicos en los que se dividió el sistema son los listados a continuación e ilustrados en la figura 6.2:

* Administración: Altas, bajas y cambios de proyectos, actividades y usuarios.
* Actualización: Sobre el estatus en actividades de desarrollo y calidad.
* Calidad: Plantillas para la implementación de distintas técnicas de detección de defectos.
* Defectos: Registro y seguimiento de defectos.
* Reportes: Generación de estadísticas y gráficas a nivel desarrollador, proyecto y empresa.

Figura 6.2. Módulos del sistema BM.

* 1. Funcionalidades y objetivos

Las funcionalidades con las que cuenta la primera versión del sistema se listan en la tabla 6.2. En ella se muestran los tipos de usuarios y lo que pueden realizar dentro del sistema. Existen tres tipos de usuario: Desarrollador, Líder de Proyecto y Gerente/Administrador.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Desarrollador** | **Líder de Proyecto** | **Gerente/Administrador** |
| Alta, baja y cambio de usuarios |  |  | X |
| Alta, baja y cambio de proyectos |  |  | X |
| Asignación de recursos humanos a proyectos |  |  | X |
| Alta, baja y cambio de actividades de desarrollo | X | X | X |
| Alta, baja y cambio de actividades de calidad | X | X | X |
| Asignación de actividades de desarrollo y calidad a miembros del equipo |  | X | X |
| Alta, baja y cambio de plantillas para actividades de calidad | X | X | X |
| Actualización de estatus de actividades de desarrollo y calidad | X | X | X |
| Registro y seguimiento de defectos | X | X | X |
| Estadísticas y gráficas a nivel usuario | X | X | X |
| Estadísticas y gráficas a nivel proyecto |  | X | X |
| Estadísticas y gráficas a nivel empresa |  | X | X |

Tabla 6.2. Funcionalidades del sistema BM.

Dentro de los objetivos trazados al momento de comenzar con la definición del **proyecto y que siguen vigentes** después de su realización, se encuentran los siguientes:

* Lograr una mejora continua en el proceso de desarrollo de software, así como una mayor calidad en el producto final.
* Reducir el costo final que conlleva implementar actividades de calidad dentro de la empresa.
* Proporcionar información valiosa, **como el costo de realización de las revisiones o el retorno de inversión de las mismas**, a la empresa para la toma de decisiones respecto a cambios en su proceso de desarrollo de software.
* Facilitar la evolución y adaptación de las diferentes actividades de aseguramiento de la calidad.

Estos objetivos se sustentan en la capacidad del sistema de proporcionar estadísticas e información que surge del registro de las actividades de desarrollo, de calidad y de los defectos. A su vez, la definición de plantillas personalizadas permite la evolución y mejora continua de las actividades de calidad, lo que a final de cuentas se traduce en una reducción de costos.

* 1. Supuestos y restricciones

Como cualquier sistema de software que se piense construir, existen supuestos y restricciones que se asumen al inicio del proyecto bajo los cuales se construye el producto final. Para el caso del BM podemos listar como supuestos los siguientes:

* Cualquier persona que utilice el sistema cuenta con las habilidades y el conocimiento necesarios para el uso del hardware y el software sobre el cual corre el sistema.
* Cualquier empresa o persona que decida utilizar el sistema contará con la infraestructura necesaria para su correcta instalación y su correcto uso.
* Cualquier empresa que decida utilizar el sistema contará con el personal adecuado para los distintos tipos de usuario del sistema, o en su caso definirá el personal que considere adecuado para ello.

Las principales restricciones para el uso correcto del sistema se encuentran a continuación:

* Al ser una aplicación Web, es necesario contar con la infraestructura de red, ya sea local o de internet, para acceder correctamente al sistema.
* El sistema en su primera versión no fue diseñado para ser visualizado en dispositivos móviles de comunicación, como por ejemplo teléfonos inteligentes. Por lo que el uso y su visualización adecuada no pueden ser garantizados al momento de acceder al sistema desde estos dispositivos.
  1. Pantallas

CAPÍTULO 7

1. Resultados Obtenidos

CAPÍTULO 8

1. Conclusiones

Referencias

|  |  |
| --- | --- |
| [1] | P. Jalote, *CMM in practice: processes for executing software projects at Infosys*, Addison-Wesley, Ed., 2000. |
| [2] | P. Johnson, "Reengineering Inspection," *Communications of the ACM.*, vol. 41, no. 2, pp. 49-52, 1998. |
| [3] | Harjumma et al, "Peer Reviews in Real Life – Motivators and Demotivators," in *Fifth International Conference on Quality Software*, 2005. |
| [4] | M. Haragopal P. Jalote, "Overcoming the NAH Syndrome for Inspection Deployment," in *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 371-378. |
| [5] | H. Lexen G. Chroust, "Software Inspections – Theorie, New approaches and an Experiment," in *Proceedings, 25th EUROMICRO Conference*, 1999. |
| [6] | Wikipedia. (2011) http://en.wikipedia.org. [Online]. <http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems> |
| [7] | P. B. Crosby, *Quality is free: The art of making quality certain*, Penguin, Ed. NY, NY, 1979. |
| [8] | J. M. Juran, *Juran on planning for quality*, Collier Macmillan, Ed. NY, NY, 1988. |
| [9] | D. Galin, *Software quality assurance: from theory to implementation*. NY: Pearson/Addison Wesley, 2004. |
| [10] | R. S. Pressman, *Software engineering: a practitioner’s approach*, McGraw-Hill, Ed. NY, 1992. |
| [11] | Collins et al, "How good is good enough?: An ethical analysis of software construction and use," *Communications of the ACM*, vol. 37, pp. 81-91, 1994. |
| [12] | J. Bach, "Good Enough Quality: Beyond the Buzzword," *IEEE Computer Society*, vol. 30, pp. 96-98, 1997. |
| [13] | Lazic et al, "The Software Quality Economics Model for Software Project Optimization," *WSEAS Transactions on Computers*, vol. 8, 2009. |
| [14] | R.E. Al-Qutaish, "An Investigation of the Weaknesses of the ISO 9126 International Standard," *Second International Conference on Computer and Electrical Engineering*, vol. 1, pp. 275-279, 2009. |
| [15] | J. Boegh, "A New Standard for Quality Requirements," *IEEE Software*, vol. 25, pp. 57-63, 2008. |
| [16] | N. Ehsan et al, "CMMI / SPICE based process improvement," in *IEEE International Conference on Management Innovation and Technology*, 2010, pp. 859-862. |
| [17] | H. M. Hosny, "The CMM software quality assurance process scaled down for small organizations," in *International Conference on Electrical, Electronic and Computer Engineering*, 2004, pp. 291-294. |
| [18] | B. W. Boehm et al, "Quantitative Evaluation of Software Quality," in *ICSE Proceedings of the Second International conference on Software Engineering*, 1976. |
| [19] | Deissenboeck et al, "Software Quality Models: Purposes, Usage Scenarios and Requirements," *ICSE Workshop on Software Quality*, pp. 9-14, 2009. |
| [20] | I. Ciordia N. Serrano, "Bugzilla, ITracker, and other Bug Trackers," *IEEE Software*, vol. 22, pp. 11-13, 2005. |
| [21] | P. F. Dubois N. Johnson, "Issue Tracking," *Computing in Science and Engineering*, vol. 5, pp. 71-77, 2003. |
| [22] | GanttProject. GanttProject Home. [Online]. <http://www.ganttproject.biz/> |
| [23] | IBM. www.ibm.com. [Online]. <http://www-01.ibm.com/software/awdtools/clearquest/> |
| [24] | S. H. Kan, *Metrics and models in software quality engineering*, Addison-Wesley, Ed., 1995. |
| [25] | N.F. Schneidewind, "Software Metrics Model for Quality Control," in *Proceedings, Fourth International Software Metrics Symposium*, 1997, pp. 127-136. |
| [26] | W. S. Humphrey, *PSP A Self-Improvement Process for Software Engineers*, Addison – Wesley, Ed. Westford, 2009. |
| [27] | K. Owens, "Software Detailed Technical Reviews: Finding and Using Defects," in *Conference Proceedings Wescon/97*, 1997. |
| [28] | B. W. Boehm, *Software engineering economics*, Prentice-Hall, Ed. NJ, 1981. |
| [29] | Laitenberger et al, "Quantitative Modeling of Software Reviews in an Industrial Setting," in *Sixth international Software Metrics Symposium*, 1999. |
| [30] | D.P. Freedman G.M. Weinberg, "Reviews, Walkthroughs, and Inspections," *IEEE Transactions on Software Engineering*, vol. SE-10, 1984. |
| [31] | Wohlin et al, "Software Inspection Benchmarking – A Qualitative and Quantitative Comparative Opportunity," in *Eighth IEEE Symposium on Software Metrics*, 2002. |
| [32] | Smart Bear Software. www.SmartBear.com. [Online]. <http://smartbear.com/PDF/11_Best_Practices_for_Peer_Code_Review.pdf> |
| [33] | Runeson et al, "What Do We Know about Defect Detection Methods?," *IEEE software*, vol. 23, pp. 82-90, 2006. |
| [34] | Laitenberger et al, "Software Inspections, Reviews & Walkthroughs," in *Proceedings of the 24rd International Conference on Software Engineering*, 2002. |
| [35] | Hall et al, "Towards Implementing Successful Software Inspections," in *Proceedings, International Conference on Software Methods and Tools*, 2000. |
| [36] | A. M. Wohaishi Y. A. Alsultanny, "Requirements of Software Quality Assurance Model," in *Second International Conference on Environmental and Computer Science*, 2009. |
| [37] | M. A. Storey et al, "TODO or to bug," in *International Conference on Software Engineering*, 2008, pp. 251-260. |

1. Apéndice A
2. Apéndice B

Vitae

Eduardo Campos Peiro nació en Guadalajara, Jalisco el 3 de noviembre de 1984. Realizó sus estudios básicos en el Colegio Liceo del Valle de esta misma ciudad. Hizo sus estudios profesionales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara, donde obtuvo el título de Ingeniero en Electrónica y Comunicaciones en diciembre del 2006. En el curso de estos estudios, realizó una estancia en el extranjero por 6 meses en la Universidad del Estado de California, en la ciudad de Chico. Durante el 2005 trabajó dentro del área de TI de Flextronics en el Campus Guadalajara. De enero de 2007 a junio de 2010 desempeñó diversos puestos en la empresa Solutia Intelligence S.A. de C.V. dentro de ellos los de desarrollador de software, líder de proyecto y gerente de proyectos. Estuvo involucrado en la implementación de diversas soluciones de radiofrecuencia y de movilidad tanto en el sector privado como público. De junio de 2010 a la fecha se desempeña como desarrollador de software dentro de IBM para el área de sistemas de almacenamiento en disco, específicamente para el análisis del desempeño de los diferentes sistemas. Cuenta con certificaciones como programador en Java, como técnico certificado para la implementación de soluciones de radiofrecuencia, con especialización en la cadena de suministro. **De Agosto de 2009 a la fecha estudia la Maestría en Ciencias de la Computación con especialidad en Ingeniería de Software en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Guadalajara.**