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

**EDUARDO CAMPOS PEIRO**

Escuela de Graduados en Ingeniería y Arquitectura

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.

TABLA DE CONTENIDO

[INTRODUCCIÓN 1](#_Toc302375962)

[1.1 Planteamiento del problema 1](#_Toc302375963)

[1.2 Objetivos y alcances 2](#_Toc302375964)

[1.3 Contribuciones de la investigación 3](#_Toc302375965)

[1.4 Organización del documento 4](#_Toc302375966)

[CALIDAD DE SOFTWARE 5](#_Toc302375967)

[2.1 ¿Qué es calidad? 5](#_Toc302375968)

[2.2 Calidad aceptable 7](#_Toc302375969)

[2.3 La importancia de la calidad 9](#_Toc302375970)

[IMPLEMENTACIÓN DE CALIDAD 10](#_Toc302375971)

[TÉCNICAS DE DETECCIÓN DE DEFECTOS 11](#_Toc302375972)

[BUG-MANAGER 12](#_Toc302375973)

[RESULTADOS OBTENIDOS 13](#_Toc302375974)

[CONCLUSIONES 14](#_Toc302375975)

[REFERENCIAS 15](#_Toc302375976)

[ANEXOS 18](#_Toc302375977)

LISTA DE TABLAS

LISTA DE FIGURAS

Capítulo 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% [10]. Además, solamente el 20% de las empresas realiza con regularidad alguna técnica de revisión [23]. 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) [14]. 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 IBM o HP, pero que no aplica en el contexto de negocio de éstas. Es decir, en general 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 [22].

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 [18].

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 [23]. 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 [24]. 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, 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, 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 una mayor calidad.

* 1. Contribuciones de la investigación

Por un lado, 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.

Por otro lado, 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 de 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 y finalmente cómo se puede mejorar paulatinamente. El capítulo 4 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 5 se explica qué es el BM, haciendo un énfasis en las técnicas de detección de defectos, y cómo se desarrolló. El capítulo 6 muestra los resultados obtenidos de la investigación teórica y del desarrollo del sistema. El capítulo 7 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 8 lista las referencias bibliográficas utilizadas durante la investigación y el capítulo 9 contiene los anexos que complementan este documento de tesis.

Capítulo 2

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 [2]. 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 [3]. 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 [5]. 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 [5].

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 “*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*” [4].

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 [25]. 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 [1]:

* 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 siguiente elementos [1]:

* 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, madurez, etc. 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 hacerlas teniendo en cuenta la restricción del costo.

* + 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 [5]:

* 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.

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.

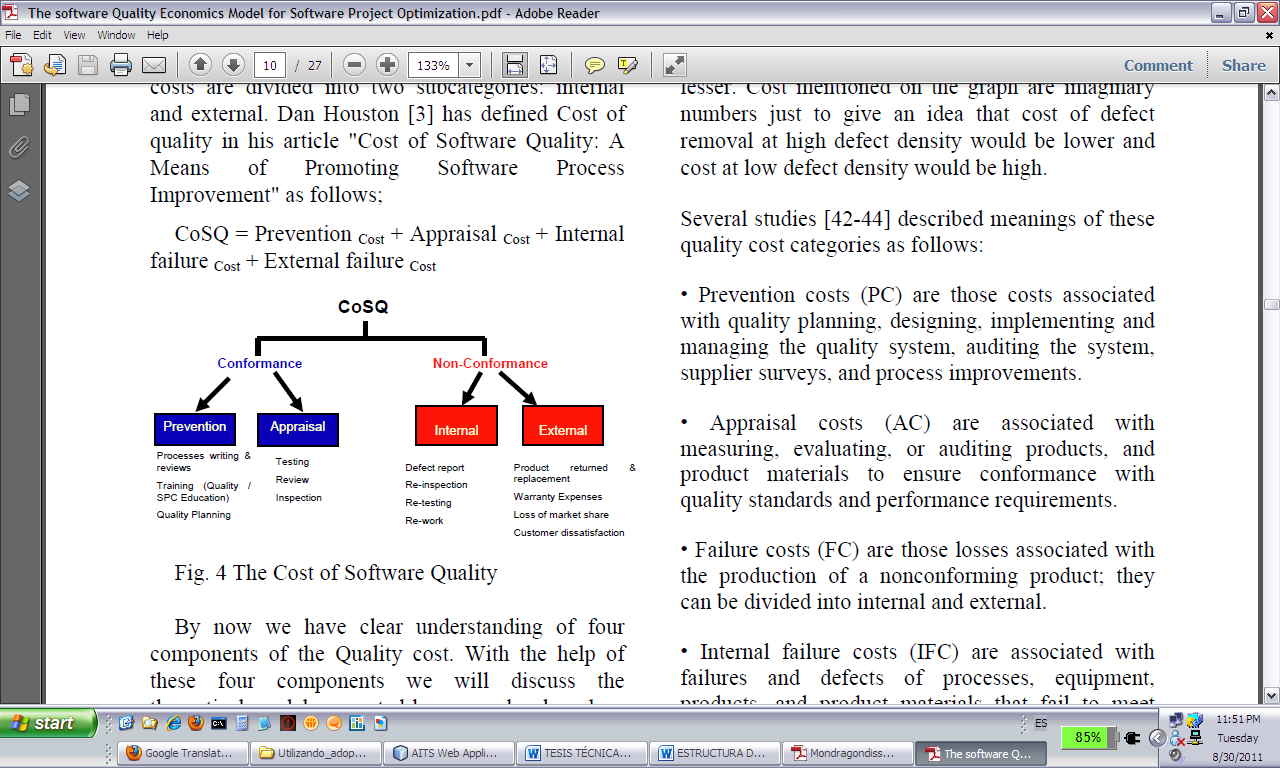


Figura 1. Costos de la calidad [9].

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 es una clara muestra de lo que se debe buscar como ideal.

Explicar lo de la tabla, lo de defect removal y relación costo-calendario.

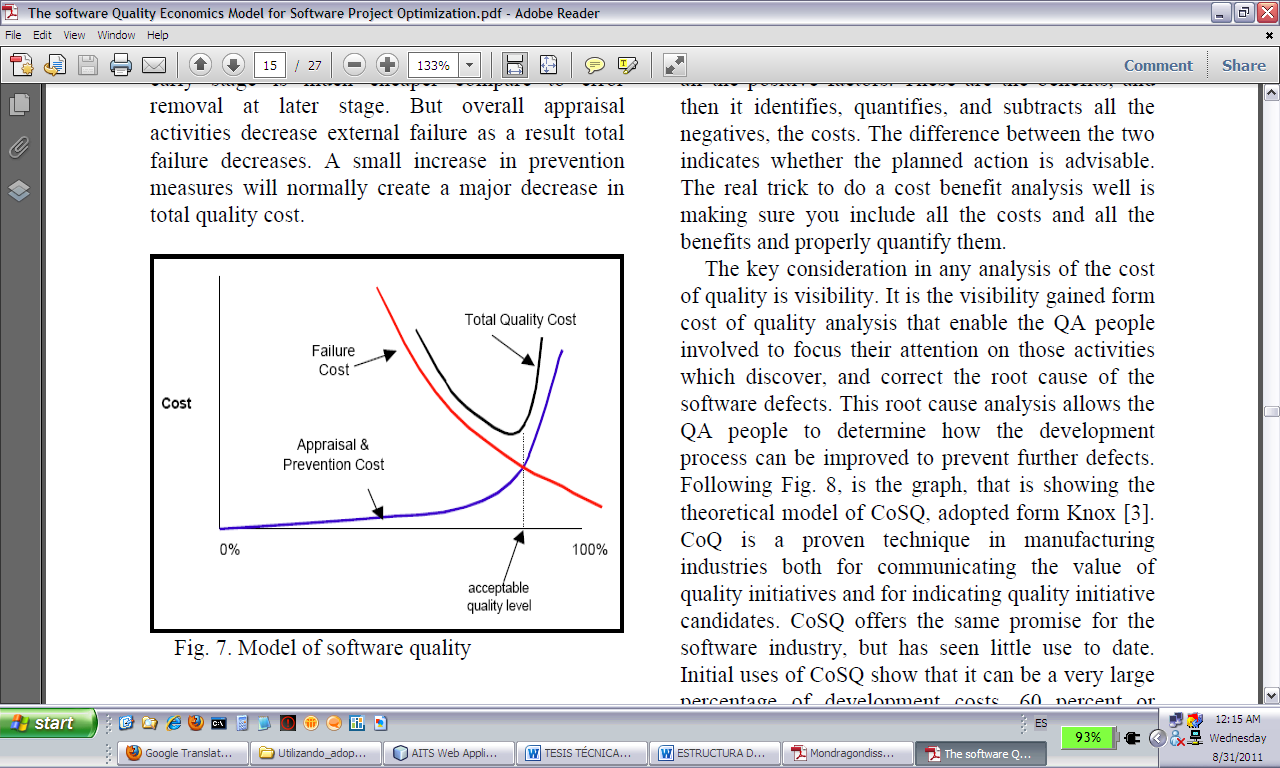


Figura 2. Modelo de calidad de software [9].

Capítulo 3

IMPLEMENTACIÓN DE CALIDAD

Capítulo 4

TÉCNICAS DE DETECCIÓN DE DEFECTOS

Capítulo 5

BUG-MANAGER

Capítulo 6

**RESULTADOS OBTENIDOS**Capítulo 7

CONCLUSIONES

Capítulo 8

REFERENCIAS

[1] J. Bach, “Good Enough Quality: Beyond the Buzzword”, *IEEE Computer Society*, vol. 30, pp. 96-98, 1997.

[2] P. B. Crosby, *Quality is free: The art of making quality certain*, NY, Penguin, 1979.

[3] J. M. Juran, *Juran on planning for quality*, NY, Collier Macmillan, 1988.

[4] R. S. Pressman, *Software engineering: a practitioner’s approach*, NY, McGraw-Hill, 1992.

[5] D. Galin, *Software quality assurance: from theory to implementation*, NY, Pearson/Addison Wesley, 2004.

[6] B. W. Boehm, *Software engineering economics*, NJ, Prentice-Hall, 1981

[7] S. H. Kan, *Metrics and models in software quality engineering*, Addison-Wesley, 1995

[8] J. Boegh, “A New Standard for Quality Requirements”, *IEEE Software*, vol. 25, pp. 57-63, 2008

[9] L. Lazic et al, “The Software Quality Economics Model for Software Project Optimization”, *WSEAS Transactions on Computers*, vol. 8, 2009.

[10] P. Jalote, *CMM in practice: processes for executing software projects at Infosys*, Addison-Wesley, 2000.

[11] W. S. Humphrey, *PSP A Self-Improvement Process for Software Engineers*, Westford, Addison – Wesley, 2009.

[12] K. Owens, “Software Detailed Technical Reviews: Finding and Using Defects”, in *Conference Proceedings Wescon/97*, 1997, doi: [10.1109/WESCON.1997.632327](http://0-dx.doi.org.millenium.itesm.mx/10.1109/WESCON.1997.632327)

[13] O. Laitenberger et al, “Quantitative Modeling of Software Reviews in an Industrial Setting”, in *Sixth international Software Metrics Symposium*, 1999, doi: [10.1109/METRIC.1999.809752](http://0-dx.doi.org.millenium.itesm.mx/10.1109/METRIC.1999.809752)

[14] L. Harjumma et al, “Peer Reviews in Real Life – Motivators and Demotivators”, in Fifth *International Conference on Quality Software*, 2005, doi: [10.1109/QSIC.2005.48](http://0-dx.doi.org.millenium.itesm.mx/10.1109/QSIC.2005.48)

[15] G.M. Weinberg and D.P. Freedman, “Reviews, Walkthroughs, and Inspections”, *IEEE Transactions on Software Engineering*, vol. SE-10, 1984.

[16] C. Wohlin et al, “Software Inspection Benchmarking – A Qualitative and Quantitative Comparative Opportunity”, in *Eighth IEEE Symposium on Software Metrics*, 2002, doi: [10.1109/METRIC.2002.1011331](http://0-dx.doi.org.millenium.itesm.mx/10.1109/METRIC.2002.1011331)

[17] O. Laitenberger et al, “Software Inspections, Reviews & Walkthroughs”, in *Proceedings of the 24rd International Conference on Software Engineering*, 2002, doi: [10.1109/ICSE.2002.146316](http://0-dx.doi.org.millenium.itesm.mx/10.1109/ICSE.2002.146316)

[18] G. Chroust and H. Lexen, “Software Inspections – Theorie, New approaches and an Experiment”, in *Proceedings, 25th EUROMICRO Conference*, 1999, doi: [10.1109/EURMIC.1999.794793](http://0-dx.doi.org.millenium.itesm.mx/10.1109/EURMIC.1999.794793)

[19] T. Hall et al, “Towards Implementing Successful Software Inspections”, in *Proceedings, International Conference on Software Methods and Tools*, 2000, doi: [10.1109/SWMT.2000.890428](http://0-dx.doi.org.millenium.itesm.mx/10.1109/SWMT.2000.890428)

[20] P. Runeson et al, “What Do We Know about Defect Detection Methods?”, in *IEEE software*, vol. 23, pp. 82-90, 2006

[21] [www.SmartBear.com](http://www.SmartBear.com), “Best Practices for Peer Code Review”.

[22] P. Jalote and M. Haragopal, "Overcoming the NAH Syndrome for Inspection Deployment", *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 371-378.

[23] P. Johnson, "Reengineering Inspection", *Communications of the ACM.* vol 41, no 2, 1998, pp.49- 52.

[24] <http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems>

[25] W. R. 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.

Capítulo 9

ANEXOS