DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMA E INFORMÁTICA

TEMA: ANÁLISIS DE HERRAMIENTAS DE DESARROLLO Y SU ADAPTACIÓN AL MODELO DE INTEGRACIÓN CONTINUA

AUTOR: VÁSCONEZ CHÁVEZ DIEGO PATRICIO

DIRECTOR: ING. MARIO RON

SANGOLQUÍ

2016

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍAS DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, "***ANÁLISIS DE HERRAMIENTAS DE DESARROLLO DE SOFTWARE Y SU ADAPTACIÓN AL MODELO DE INTEGRACIÓN CONTINUA***" realizado por el señor ***DIEGO PATRICIO VÁSCONEZ CHÁVEZ***, ha sido revisado en su totalidad y analizado por el software anti-plagio, el mismo cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, por lo tanto me permito acreditarlo y autorizar al señor ***DIEGO PATRICIO VÁSCONEZ CHÁVEZ*** para que lo sustente públicamente.

Sangolquí, 23 de febrero del 2016

ING. MARIO BERNABÉ RON EGAS

DIRECTOR

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORÍA DE RESPONSABILIDAD

Yo, ***DIEGO PATRICIO VÁSCONEZ CHÁVEZ***, con cédula de identidad N° 170885644-6, declaro que este trabajo de titulación "***ANÁLISIS DE HERRAMIENTAS DE DESARROLLO DE SOFTWARE Y SU ADECUACIÓN AL MODELO INFORMÁTICO DE INTEGRACIÓN CONTINUA***" ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaro que este trabajo es de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 23 de febrero del 2016

DIEGO PATRICIO VÁSCONEZ CHÁVEZ

C.C. 170885644-6

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN

Yo, ***DIEGO PATRICIO VÁSCONEZ CHÁVEZ***, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar en la biblioteca Virtual de la institución el presente trabajo de titulación **"ANÁLISIS DE HERRAMIENTAS DE DESARROLLO DE SOFTWARE Y SU ADECUACIÓN AL MODELO INFORMÁTICO DE INTEGRACIÓN CONTINUA"** cuyo contenido, ideas y criterios son de mi autoría y responsabilidad.

Sangolquí, 23 de febrero del 2016

DIEGO PATRICIO VÁSCONEZ CHÁVEZ

C.C. 170885644-6

DEDICATORIA

A mi familia que ha sabido apoyar el esfuerzo dedicado a este proyecto y a mis padres, con quienes hubiese querido compartir el logro alcanzado.

Diego Vásconez

AGRADECIMIENTO

Agradezco a mi esposa y a mis hijos por su comprensión durante el tiempo dedicado a este proyecto.

Al Ing. Mario Ron, Director de Tesis por todos sus conocimientos académicos y colaboración para poder culminar este proyecto.

Diego Vásconez

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El Banco Central del Ecuador – BCE, produce sistemas por medio de los cuales ofrece servicios tanto para el Sistema Financiero Nacional, las Instituciones del Sector Público y el país en general. Estos servicios son presentados a través de redes privadas y por la Internet haciendo necesario que, las aplicaciones desarrolladas requieran de nuevas características y mayores niveles de calidad en comparación con las que fueron construidas para uso interno con la arquitectura Cliente-Servidor. Las prácticas planteadas en la Integración Continua prometen contribuir a elevar la calidad de los sistemas desarrollados utilizando dichas prácticas es por eso que, como objetivo de este trabajo se analiza herramientas de desarrollo y su adecuación a las prácticas del modelo de Integración Continua. Para lograr este objetivo se realiza una evaluación de la situación actual determinando, que requerimientos deben cubrir las herramientas disponibles en el mercado, estableciendo una arquitectura candidata considerando que, en el BCE ya se cuenta con un Sistema de Control de Versiones, un Gestor de Librerías y una herramienta para la Construcción Automática de Software para obtener como resultado el producto mejor puntuado usando los parámetros y métricas establecidos para su evaluación. Se concluye que, las prácticas propuestas en el modelo de Integración Continua sumada a una infraestructura que automatice ciertas tareas de la producción de software, permite la definición de un marco de referencia para mejorar la realización de pruebas y validad los productos desarrollados.

PALABRAS CLAVE:
CALIDAD
SERVICIO
DESARROLLO
HERRAMIENTAS
AUTOMATIZACIÓN

ABSTRACT

The Central Bank of Ecuador - ECB produces systems through which offers services for both the National Financial System Institutions Public Sector and the country in general. These services are presented through private networks and the Internet making it necessary for the developed applications require new features and higher levels of quality compared to those that were built for internal use with client-server architecture. Practices raised in the Continuous Integration promise to help raise the quality of the systems developed using these practices is why, as an objective of this work development tools and matching practices Continuous Integration model is analyzed. To achieve this objective an assessment of the current situation is done by determining which requirements should cover the tools available in the market, establishing an architecture candidate whereas the ECB already has a System Version Control a library manager and a tool for Automatic Software Construction to result in the highest rated product using established parameters and metrics for evaluation. It is concluded that the practices proposed in the model of Continuous Integration together with an infrastructure that automates certain tasks of production software, allows the definition of a framework to improve testing and validate the products developed

KEYWORDS:
QUALITY
SERVICE
DEVELOPMENT
TOOLS
AUTOMATION

CAPÍTULO 1

1 PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

1.1 Introducción

La construcción de un producto de software es un ejercicio de trabajo y tiempo considerable, de gran complejidad incluso en sus formas más simples y que debe buscar que dicho producto permanezca activo en el tiempo considerando que el software en sí tiene una importancia cada vez más grande en la sociedad actual.

Por ello el desarrollo de aplicaciones informáticas por medio de metodologías tradicionales, no solamente que puede plantearse como algo negativo, sino que implica riesgos inasumibles para las empresas que se dedican a esta actividad, aunque demasiadas de ellas aún no le dan la importancia que tiene.

Así, al menos desde la década de los años 70 del siglo pasado, ha ido evolucionando la idea de la Gestión del Ciclo de Vida de las Aplicaciones (ALM[1]), que contempla las fases que todo proyecto deberá superar, para intentar conseguir ese objetivo tan resbaladizo como es el éxito.

En el presente proyecto de tesis, se analizarán herramientas de Integración Continua para responder a las necesidades de cualquier metodología que utilicemos para gestionar el ciclo de vida de nuestras aplicaciones, con un importante componente de la filosofía asociada a las metodologías ágiles de desarrollo de software.

1.2 Descripción general de la gestión del ciclo de vida de aplicaciones

El Ciclo de Vida de una aplicación (ALM) se puede dividir a grandes rasgos en las siguientes agrupaciones:

Análisis: Se realiza el análisis de los requisitos que conforman las funcionalidades esperadas por los usuarios finales del sistema.

[1] Application Lifecycle Management (Gestión del Ciclo de Vida de las Aplicaciones)

Diseño: Definición de la arquitectura de los diferentes componentes que constituyen la aplicación. Como puede ser el esquema de datos, la capa de negocio o el interfaz de usuario.

Codificación: Lo que hacemos todos los días. Digitar el código de la aplicación y realizar su construcción.

Pruebas: Preparar y ejecutar las pruebas necesarias para asegurar que la aplicación realiza las tareas esperadas y devuelve las respuestas correctas. En resumen, que sea útil para sus usuarios.

Documentación: Trata sobre transmitir el conocimiento generado durante todo el Ciclo de Vida del software, a actores que no han participado en la construcción o que lo han hecho pero en otra fase o en otro momento temporal y a futuros actores que requieran del conocimiento producido. Es de señalar que la documentación es la base de la gestión de los proyectos.

1.3 Algunos tipos de paradigmas

Partiendo de estas fases tan genéricas, nos encontramos con que se han desarrollado diferentes tipos de metodologías para definir cuándo, cuánto y cómo se realiza la construcción de las aplicaciones. A continuación, se describirán las más importantes de forma resumida.

1.3.1 Desarrollo en Cascada

Como indica su nombre, las fases se van cumplimentando una detrás de otra (en algunas de las implementaciones esto no es necesariamente así) esperando el final de una para iniciar la siguiente. Es con mucho la metodología más utilizada ya que da una sensación muy importante de previsión y robustez en los resultados, que a su vez es su máxima debilidad considerando que el desarrollo de software ha demostrado ser todo menos previsible. Además son muy costosos los cambios en estadios finales de la construcción.

El nexo de unión entre las fases es la documentación.

1.3.2 Desarrollo en Espiral

El núcleo de esta forma de construir software es el riesgo inherente que todo desarrollo implica. Dependiendo del riesgo detectado, así se define los ciclos o

iteraciones de trabajo que se van a realizar. Es una metodología robusta orientada a grandes y complejos proyectos pero con una capacidad pobre de predicción.

Requiere personal expertos en gestión de riesgos.

1.3.3 Desarrollo iterativo y creciente

Partiendo de la realidad de los inconvenientes de la metodología en cascada, se define este tipo de construcción en donde se parte de una implementación simple de los requerimientos del sistema para ir evolucionando iterativamente la aplicación hasta su implementación completa. Esta forma de desarrollar es tolerante a los cambios en cualquier momento del ciclo de vida pero tiene el inconveniente que es poco predictiva y puede llegar a ser confusa su gestión.

Requiere un cliente o usuario final muy comprometido y dispuesto a implicarse plenamente en tiempo y esfuerzo junto con el equipo de desarrollo.

Hay una multitud de metodologías que extraen lo que consideran lo mejor de cada una de las otras para componer formas de Ciclos de Vida más ajustados a las ideas o necesidades de diferentes autores. Además tenemos que bajar al nivel de la implementación y entonces entramos en una miríada de acrónimos que indican los diferentes sabores que se proponen como son RUP[2], Scrum[3], XP[4], DSDM[5], RAP[6], Metrica 3[7], PMI[8], Prince 2[9], entre otros.

Es necesario mantenerse a un nivel de ingeniería de software para que se perciba que somos una industria muy joven que aún está buscando, en sus primeros balbuceos, la forma de realizar de manera ordenada la unión de un proceso industrial con los procesos intelectuales y artísticos que implica la programación de sistemas informáticos.

---

[2] Proceso Unificado de Rational o Rational Unified Process en inglés

[3] Marco de trabajo por el cual las personas pueden acometer problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente. Es ligero, fácil de comprender y extremadamente difícil de llegar a dominar. (SCHWABER, Ken; SUTHERLAND, Jeff;, 2013)

[4] Programación extrema o eXtreme Programming en inglés

[5] Método de desarrollo de sistemas dinámicos o Dynamic Systems Development Method en inglés

[6] Real Agility Program en inglés

[7] Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información, versión 3. (PAE, 2001)

[8] Instituto de Gestión de Proyectos o Project Management Institute en inglés

[9] Acrónimo para PRojects IN Controlled Environments, estándar de facto basado en procesos para una efectiva gestión de proyectos. (OGC, 1996)

1.3.4 Integración Continua

En un equipo de desarrollo software la división de trabajo es una práctica normal y cada programador se dedica al desarrollo de una tarea y un código específico, el cual es una pieza que posteriormente tiene que encajar en el rompecabezas del producto final. Por tanto, el trabajo de encajar esas piezas de la mejor forma posible es una tarea en sí complicada e importante en el desarrollo de un proyecto software. En este aspecto y muy familiarizado con el desarrollo ágil tiene cabida la Integración Continua.

La Integración Continua es definida por Martin Fowler (FOWLER, 2006), uno de los padres de esta, como:

"Práctica de desarrollo de software donde los miembros de un equipo integran su trabajo frecuentemente, por lo menos diariamente y llegando a múltiples integraciones al día. Cada integración se comprueba por una construcción[10] automatizada para detectar errores de integración lo antes posible. Muchos equipos encuentran esta aproximación como una forma de reducir significativamente problemas de integración y que permite desarrollar cohesivamente software de forma más rápida."

1.4 Planteamiento del Problema

Hacia la segunda mitad de la década de los 90, en ese entonces la Dirección de Informática[11] del Banco Central del Ecuador (BCE) deja atrás los equipos Main

[10] Building en inglés, cuyo significado comprende la organización, construcción y compilación de código.

[11] De acuerdo al "Estatuto Orgánico de Gestión Organizacional por Procesos del Banco Central del Ecuador" la anterior Dirección de Informática ahora es la Coordinación General de Tecnologías de Información y Comunicación.

Frame[12] de IBM[13] y el desarrollo de aplicaciones basadas en el lenguaje de programación Cobol[14] CICS[15].

En esa misma década, la introducción de los computadores personales y la tendencia a incrementar su capacidad de procesamiento promovió la adopción de lenguajes de programación tipo dBase[16], FoxPro[17] o Clipper[18] para desarrollar aplicaciones independientes al MainFrame y que podían ser manejadas directamente por el usuario pero limitadas al ámbito de su equipo personal aunque, en un futuro cercano se aprovecharon las capacidades de las incipientes redes de comunicación de la época.

---

[12] Mainframe es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

[13] IBM International Business Machines Corp. es una empresa multinacional estadounidense de tecnología y consultoría que fabrica y comercializa hardware y software para computadoras, ofrece servicios de infraestructura, alojamiento de Internet, y consultoría en una amplia gama de áreas relacionadas con la informática, desde computadoras centrales hasta nanotecnología.

[14] COBOL del inglés COmmon Business-Oriented Language o, Lenguaje Común Orientado a Negocios fue creado en el año 1959 con el objetivo de crear un lenguaje de programación universal que pudiera ser usado en cualquier ordenador, ya que en los años 1960 existían numerosos modelos de ordenadores incompatibles entre sí, y que estuviera orientado principalmente a los negocios, es decir, a la llamada informática de gestión.

[15] CICS, acrónimo en inglés de Customer Information Control System (en español, Sistema de control de información de clientes), es un gestor transaccional, o monitor de teleproceso, que se ejecuta principalmente en mainframes IBM con los sistemas operativos OS/390, z/OS o VSE. También existen versiones de CICS para otros entornos, como OS/400, OS/2.

[16] dBASE fue el primer sistema de gestión de base de datos usado ampliamente para microcomputadoras, publicado por Ashton-Tate para CP/M, y más tarde para Apple II, Apple Macintosh, UNIX [1], VMS [2], e IBM PC bajo DOS donde con su legendaria versión III Plus se convirtió en uno de los títulos de software más vendidos durante un buen número de años.
dBASE nunca pudo superar exitosamente la transición a Microsoft Windows y terminó siendo desplazado por otros productos como Paradox, Clipper, y FoxPro.

[17] FoxPro (acrónimo de FoxBASE Professional) es un lenguaje de programación orientado a procedimientos (procedures), a la vez que un Sistema Gestor de Bases de datos o Database Management System (DBMS), publicado originalmente por Fox Software y posteriormente por Microsoft, para los sistemas operativos MS-DOS, MS Windows, Mac OS y UNIX.

[18] Clipper es un lenguaje de programación procedural e imperativo creado en 1985 por Nantucket Corporation y vendido posteriormente a Computer Associates, la que lo comercializó como CA-Clipper. En un principio Clipper se creó como un compilador para el sistema gestor intérprete de bases de datos dBase III, pero con el tiempo el producto evolucionó y maduró, convirtiéndose en un lenguaje compilado más poderoso que el original, no sólo por sus propias implementaciones sino también por las ampliaciones desarrolladas por terceros en C, Ensamblador y Pascal, de los que fue heredando características. Esto lo convirtió en la herramienta líder de desarrollo de aplicaciones de bases de datos relacionales bajo sistema operativo MS-DOS, sobre todo programas de gestión, contabilidad y facturación.

La opción escogida para reemplazar a los Mainframe fueron equipos con arquitectura RISC[19] de Sun Microsystems[20] con el sistema operativo Solaris[21] y sobre el cual se incorpora el motor de base de datos Sybase ASE[22]. La herramienta de desarrollo seleccionada para explotar la información almacenada en la nueva base de datos sería PowerBuilder[23], dirigida al Desarrollo Rápido de Aplicaciones y a la programación orientada a eventos que venía imponiéndose con la aparición de sistemas operativos con interfaces gráficas como Windows. Sobre esta nueva infraestructura y con las nuevas herramientas de desarrollo se emprende en un agresivo proceso de construcción de aplicaciones Cliente / Servidor en las que se intenta aprovechar tanto las capacidades de procesamiento de los equipos personales como de los servidores de base de datos.

Paralelamente a todo este proceso, se estaría dando la evolución de Internet y un sin número de tecnologías orientadas a presentar servicios a través de ella. Es así que en el año 2002, el Banco Central del Ecuador inicia la entrega de servicios a través de tecnologías Web usando una red privada dirigida al Sistema Financiero Nacional y por la cual se accede al Sistema Nacional de Pagos.

Pero el lograr que este producto llegue a producción no sería una cosa fácil. Uno de los primeros problemas que se pueden identificar es la gran cantidad de tecnologías que se deben dominar, no basta con el conocimiento del lenguaje de programación establecido. Para crear una página web medianamente útil hay que

---

[19] RISC, en inglés Reduced Instruction Set Computer o en español, Computador con Conjunto de Instrucciones Reducidas, es un tipo de arquitectura de microprocesador que utiliza un conjunto pequeño, altamente optimizado de instrucciones, en vez de un conjunto más especializado de instrucciones a menudo encontrado en otros tipos de arquitecturas.

[20] Sun Microsystems fue una empresa informática que se dedicaba a vender estaciones de trabajo, servidores, componentes informáticos, software (sistemas operativos) y servicios informáticos. Fue adquirida en el año 2010 por Oracle Corporation, y formó parte de los iconos de Silicon Valley, como fabricante de semiconductores y software.

[21] Solaris es un sistema operativo de tipo Unix desarrollado desde 1992 inicialmente por Sun Microsystems y actualmente por Oracle Corporation como sucesor de SunOS. Es un sistema certificado oficialmente como versión de Unix. Funciona en arquitecturas SPARC y x86 para servidores y estaciones de trabajo.

[22] ASE en inglés Adaptive Server Enterprise es el motor de bases de datos (RDBMS) insignia de la compañía Sybase. ASE es un sistema de gestión de datos, altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo.

[23] SAP Sybase PowerBuilder es la herramienta más productiva para la construcción de aplicaciones de negocio dirigidas a datos. Su tecnología patentada DataWindow sigue siendo la manera más rápida y eficaz para acceder a conjuntos de datos complejos, aplicar la lógica de negocio, y mostrar datos con sólo unas pocas líneas de código.

conocer de HTML[24], CSS[25], Javascript[26], controles en el cliente, controles en el servidor, validaciones en el cliente, validaciones en el servidor, XML[27], XSLT[28], y las cosas pueden empeorar si hay que integrar algún applet java, o peor, flash. El esquema simple de request-response no es seguido por muchas aplicaciones web debido a que no mantiene estado. Hay que inventar técnicas para mantener el estado de la aplicación, sesiones, variables de sesión, cookies, estado de la vista, cache de aplicaciones entre otras.

El desarrollo de aplicaciones web se configura como un nuevo escenario para la generación de productos del BCE. Se adopta Java como plataforma y principal lenguaje de programación, esto introduce la idea de programar aplicaciones para ser ejecutadas en un servidor a diferencia de lo que se venía haciendo con aplicaciones dirigidas a ejecutarse en las estaciones de trabajo de los usuarios. Se usa el lenguaje de programación para el servidor para escribir instrucciones que serán interpretadas por el navegador de una máquina cliente. Se impone un cambio en la forma de programar pasando de, lenguajes en los que se había consolidado la programación estructurada para entrar a trabajar con un lenguaje de programación orientado a objetos.

Se hacen necesarios conocimientos más profundos sobre protocolos de comunicaciones como HTTP, HTTPS, TCP, IP, SMTP, ICMP entre otros.

La plataforma java y sus especificaciones permite que se desarrollen un sin número de productos orientados a soportar la forma de construir aplicaciones que establecía su propuesta. Esto si bien resulta en una ventaja por la gran cantidad de alternativas que en un momento determinado se puede tener a disposición, por otro

---

[24] HTML siglas de HyperText Markup Language en inglés, en español Lenguaje de Marcas de Hipertexto, hace referencia al lenguaje de marcado para la elaboración de páginas web. Define una estructura básica y un código denominado código HTML para la definición de contenido de una página web, como texto, imágenes, entre otros.

[25] CSS en inglés Cascading Style Sheets es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

[26] JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

[27] XMLen inglés Extensible Markup Language, en español Lenguaje de Marcado Extensible, es un formato de texto simple, muy flexible. Originalmente diseñado para cumplir con los desafíos de la edición electrónica a gran escala, también está desempeñando un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y en otras partes.

[28] XSL en inglés Extensible Stylesheet Language, en español Lenguaje Extensible de estilos, es una familia de recomendaciones para la definición de transformación de documentos XML y presentación.

lado, termina volviéndose un problema puesto que, se debe decidir con que herramienta o herramientas se estructuraría una arquitectura sobre las cual trabajen las aplicaciones que se construyan.

Para introducir el paradigma Cliente / Servidor en el proceso de desarrollo se invirtieron suficientes recursos para capacitar al personal, contratar programadores experimentados, acceder a un acompañamiento del proveedor de la herramienta de desarrollo. En contraste, la adopción de las tecnologías web presentó un empuje limitado debido principalmente al limitado número de aplicaciones a desarrollarse para el Internet en un inicio y, a una relativa escases de programadores experimentados en un lenguaje relativamente nuevo como Java y con un mayor de nivel de dificultad comparado con los lenguajes de programación que se habían utilizado hasta entonces. Además de un buen conocimiento del lenguaje de programación, se ve la necesidad de entender cómo aprovechar las denominadas APIs[29] como por ejemplo: JTA Java Transaction API, JAXP para procesar archivos XML, JMS Java Message Service, API para el manejo de mensajería, entre muchas otras.

Otro de los inconvenientes a considerar es la rotación de personal debido, a los múltiples procesos de reducción por los que habría atravesado el ex Instituto Emisor desde la década de los noventa y a limitaciones establecidas para no incrementar la masa salarial. Esta situación provoca y ha provocado debilidad y riesgo en la conformación de los equipos de trabajo ya que, se han presentado ocasiones en las que luego de capacitarse y lograr un buen grado de experiencia, los programadores a contrato se han separado de la institución en busca de estabilidad o mejores condiciones de trabajo dejando desarmados los proyectos a nivel del recurso humano.

El hecho es que el desarrollo de aplicaciones web para entregar servicios a través de la Internet implica tomar en cuenta un gran número de variables que anteriormente no eran necesario considerarlas para el desarrollo de las aplicaciones Cliente / Servidor. La seguridad, la latencia provocada por los elementos de comunicaciones, la infraestructura de servidores, la compatibilidad con los

[29] API en inglés Application Program Interface, en español Interfaz de Programación de Aplicaciones, es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

navegadores de los usuarios y el tiempo de respuesta, son algunos ejemplos de elementos que se introducen en este nuevo escenario.

Con los antecedentes planteados, se puede entender que los productos entregados al área de producción para el inicio de sus operaciones muchas veces presentan problemas que son identificadas en el día a día de un ambiente de trabajo real y, los usuarios finales terminan siendo quienes evidencian dichos problemas revirtiendo estas novedades en quejas dirigidas hacia las autoridades de turno.

Es así que desde la DIO[30] se ve la necesidad de buscar mecanismos de mejorar la calidad de los productos de software desarrollados en el BCE utilizando para ello una infraestructura encaminada a soportar el proceso de desarrollo de software. Esta infraestructura estaría constituida por un conjunto de herramientas que conformarían un ecosistema con el cual estarían interactuando los equipos de trabajo participantes, a saber, en la Coordinación General de Tecnologías de la Información y Comunicaciones (CGTIC) tendríamos: Desarrollo de Software, Gestión de Proyectos, Gestión de Calidad, Seguridades Informáticas e Infraestructura y Operaciones.

1.5 Contextualización del Problema

Este proyecto de tesis se desarrollará en el Banco Central del Ecuador dentro de lo que ahora es la Coordinación General de Tecnologías de la Información y Comunicación y específicamente en la Dirección de Infraestructura y Operación de TI.

El ex Instituto emisor ha ido incorporando a sus funciones, nuevos servicios basados en tecnología entre ellos, el Sistema Nacional de Pagos, Depósito Centralizado de Valores, la Cámara Electrónica de Compensación de Cheques y para el tiempo en que se desarrolla este trabajo estaría iniciando sus operaciones el proyecto de Dinero Electrónico.

De acuerdo con la "Ley Reformatoria a la Ley de Régimen Monetario y Banco del Estado"

"El Banco Central del Ecuador es una persona jurídica de derecho público, de duración indefinida, es responsable de su gestión técnica y administrativa y con patrimonio propio. Tendrá como funciones instrumentar, ejecutar, controlar y aplicar

[30] Dirección de Infraestructura y Operaciones de TI.

las políticas monetaria, financiera, crediticia y cambiaria del Estado y, como objetivo velar por la estabilidad de la moneda. Su organización, funciones y atribuciones, se rigen por la Constitución, las Leyes, su Estatuto y los reglamentos internos, así como por las regulaciones y resoluciones que dicte su Directorio, en materias correspondientes a política monetaria, financiera, crediticia y cambiaria del país. En su administración interna deberá aplicar la leyes y normas vigentes para el sector público."

La figura 1 muestra el organigrama del Banco Central del Ecuador en el cual se identifica a la CGTIC.

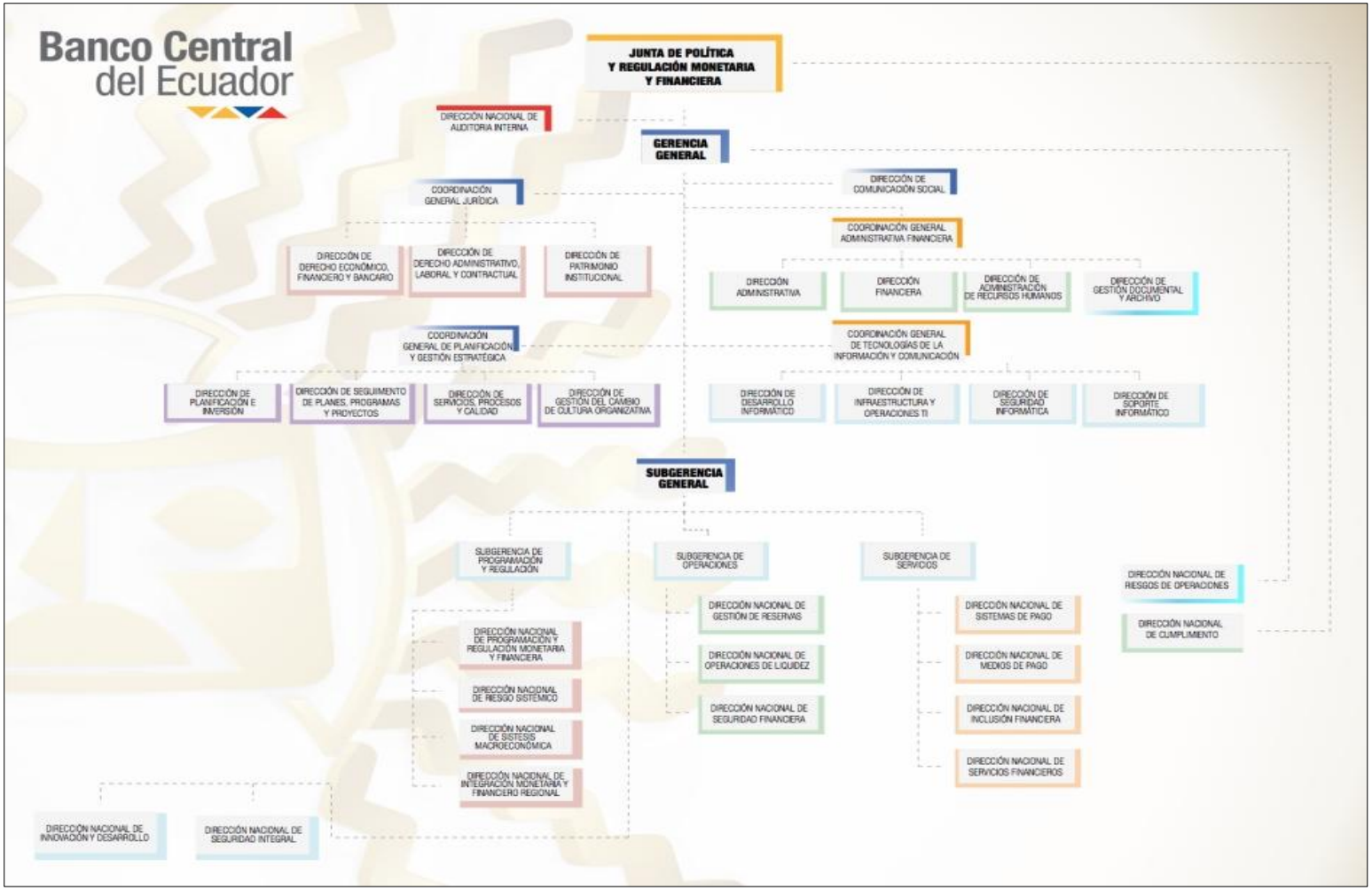


Figura 1: Organigrama del Banco Central del Ecuador.
Fuente: http://www.bce.fin.ec

1.6 Justificación

El hecho de que lleguen a producción sistemas que fueron desarrollados siguiendo el ciclo de vida en cascada contribuye a que las pruebas del producto se realicen en la etapa previa al inicio de sus operaciones. El escenario habitual que se presenta es que, el tiempo planificado para las pruebas en comparación con el tiempo dedicado al desarrollo es demasiado corto, sumado a esto, las constantes presiones que suelen presentarse por el cumplimiento de plazos y fechas comprometidas con anterioridad provoca que las pruebas se dediquen fundamentalmente, a verificar el cumplimiento de los requerimientos funcionales muchas veces, dejando de lado la evaluación de otras cualidades de los productos de software.

El proceso de control de calidad se fundamenta en pruebas de caja negra basándose en un plan de pruebas establecido por el equipo desarrollador y después de una revisión previa realizada por el usuario responsable de la aplicación o sistema.

No se realizan o se realizan pocos controles del código de la aplicación, debido a que, la evaluación del código elaborado por un programador normalmente requiere de otro programador de igual o mayores conocimientos y experiencia. El equipo de Control de Calidad no necesariamente cuenta con las características técnicas para realizar este tipo de validación por lo que herramientas como las que se incorporarían con la Integración Continua facilitarían esta tarea.

Entre otras cosas, la introducción de las prácticas de Integración Continua está orientada a automatizar varias pruebas que muchas veces no han sido valoradas y que permiten que la aplicación sea probada cada vez que se realiza un cambio sobre el código de la misma.

1.7 Objetivos

1.7.1 Objetivo General

Analizar herramientas de desarrollo de software y su adecuación al modelo informático de Integración Continua.

1.7.2 Objetivos Específicos

1. Identificar metodologías, mecanismos y práctica que permitan mejorar la calidad del software del BCE.
2. Identificar herramientas dirigidas a implementar el modelo de Integración Continua.
3. Determinar las mejores herramientas para una eventual incorporación a la infraestructura de desarrollo del BCE.

1.8 Alcance

Realizar una propuesta de buenas prácticas que se incorporen al proceso de desarrollo de software que actualmente viene aplicándose en el BCE. Estás buenas prácticas serán soportadas por un conjunto de herramientas de Integración Continua.

El resultado de este trabajo será la selección de los productos que cumplan de la mejor manera los requerimientos establecidos para el soporte de las diferentes actividades a realizarse en el ciclo de vida de un proyecto de software y de acuerdo a las recomendaciones que se adopten de las metodologías ágiles de desarrollo y de la misma Integración Continua.

1.9 Metodología

1.9.1 Marco Teórico

Conjunto de enunciados y elementos conceptuales que sustentan la investigación realizada, contribuyen a establecer y abordar el problema que se está tratando.

1.9.2 Análisis de la Situación actual y Determinación de Requerimientos

Identificación de las razones que impulsan el desarrollo de esta tesis. Identificación de falencias en el proceso de desarrollo de software en el BCE, generación de nuevas versiones de productos que ya se encuentra en producción y características de calidad de los mismos.

1.9.3 Análisis de las herramientas disponibles en el mercado

Es necesario evaluar la oferta existente tanto a nivel de productos propietarios como los de uso gratuito para determinar las mejores alternativas que se adapten a la realidad del proceso de desarrollo de software que se lleva al momento en el BCE y con la perspectiva de aportes y contribuciones al mismo.

1.9.4 Estudio o diseño de la arquitectura candidata

Una parte muy importante es cómo utilizar adecuadamente las herramientas de la arquitectura para conseguir el resultado esperado en la posterior consecución de un proyecto de desarrollo. Para este cometido es esencial realizar el diseño adecuado de la arquitectura en función de las herramientas analizadas para el desarrollo en un entorno ágil.

1.9.5 Selección de las herramientas

El presente trabajo se apoyará en una guía para la selección de software basada en estándares de calidad de la cual, se hará una descripción de sus partes y nos basaremos en ella en lo que es el Proceso de Evaluación de Software para, escoger los productos aplicando los siguientes pasos:

- Establecer el propósito de la evaluación
- Identificar el tipo de producto
- Especificar el Modelo de Calidad
- Seleccionar métricas
- Establecer niveles, escalas para las métricas

- Establecer criterios de valoración
- Tomar medidas
- Comparar con los criterios
- Valorar resultados
- Documentación

1.9.6 Elaboración del Informe de Resultados

Presentación de reporte elaborado en base del análisis de las herramientas seleccionadas considerando su aporte para la adopción de las prácticas de la Integración Continua y su incorporación al proceso de desarrollo de software.

1.10 Herramientas

1.10.1 Hardware

- Computador de Escritorio con las herramientas necesarias para el desarrollo del proyecto
- Conexión a internet de al menos 512 Kbps
- Equipos Oracle de tecnología SPARC propiedad del BCE
- Red de comunicaciones para acceder e integrar diferentes elementos de la infraestructura

1.10.2 Software

- Virtual Box para la virtualización de entornos en donde se puedan probar los productos analizados para plataformas x86.
- Oracle Virtual Machine - OVM para la generación de entornos virtuales sobre plataformas SPARC para la prueba de software
- Sybase Adaptive Server Enterprise: Motor de gestión base de datos relacional utilizada como estándar en el BCE.
- Glassfish: Servidor de aplicaciones utilizado como plataforma para la instalación del runtime de los diferentes buses de servicio

- Microsoft Project para la elaboración y seguimiento del plan de trabajo del proyecto
- Sistema Operativo Oracle Solaris como plataforma de prueba de los productos evaluados
- Sistema Operativo Red Hat Linux como alternativa de plataforma
- Herramientas ofimáticas para la elaboración de presentaciones, informes y cuadros.

1.11 Factibilidad

1.11.1 Factibilidad Operativa

Para el desarrollo del proyecto se cuenta con el apoyo de la Dirección de Infraestructura y Operaciones de TI del Banco Central del Ecuador como auspiciante de este proyecto de tesis la cual, facilitará toda la información y la infraestructura que se requiera y colaborará con todo lo necesario para la satisfactoria culminación del mismo.

1.11.2 Factibilidad Técnica

El proyecto es factible en cuanto ya se cuenta con una experiencia previa en la instalación, configuración de ciertos elementos que componen el Ecosistema de Desarrollo de Software. Se cuenta con equipos y software en el que actualmente se opera con varios productos que podría considerarse un avance hacia la consecución de los objetivos de este proyecto aunque es necesario que se actualicen las versiones de los mismos.

1.11.3 Factibilidad Económica

Para el desarrollo del presente proyecto no se contemplará al costo de los productos como una limitación ya que se trabajará con demos o versiones de prueba o entrenamiento por lo tanto, teóricamente se considerará un presupuesto infinito. El tesista asumirá los costos de desarrollo del proyecto de tesis. La tabla 1 detalla el presupuesto estimado para la ejecución del presente proyecto de tesis.

Tabla 1
Estimación del presupuesto del proyecto de tesis

PRESUPESTO PROYECTO DE TESIS			
DESCRIPCION	**VALOR**	**MESES**	**SUBTOTAL**
Hardware			
Equipo de Computo Personal	1,170		1,170
Impresora	70		70
UPS	35		35
Cartuchos de Tinta	45		45
Equipos facilitados por el BCE	0		0
Infraestructura de Red facilitada por el BCE	0		0
Software			
Virtual Box	0		0
Oracle Virtual Machine	0		0
Oracle Solaris 10	0		0
Mozilla Firefox	0		0
Microsoft Word estándar del BCE	0		0
Microsoft Excel estándar del BCE	0		0
Microsoft PowerPoint estándar del BCE	0		0
Microsoft Project estándar del BCE	0		0
Recurso Humano			
Administrador de Capa Media (Middleware)	1,200	7	8,400

CAPÍTULO 2

2 MARCO TEÓRICO

2.1 Integración Continua

"La Integración Continua es una práctica de desarrollo de software en la que los diferentes miembros de un equipo de desarrollo de software integran frecuentemente su trabajo para obtener un sistema completo o parcialmente completo. Cada integración es verificada con una construcción automática del software acompañada generalmente, de pruebas para detectar errores tan rápido como sea posible. Además, se generan informes para mostrar el resultado de cada integración.

Las principales ventajas de esta práctica son la reducción de problemas de integración y el desarrollo de un software con una mayor cohesión y, por tanto, con una mayor calidad y menores riesgos." (FOWLER, 2006)

El objetivo de este capítulo es introducir qué es y para qué se utiliza la práctica de la Integración Continua en los desarrollos de software. Se analizan las ventajas y los inconvenientes, así como las mejores prácticas recomendadas para implantar su uso. Por último, se estudian algunas de las herramientas que actualmente se utilizan para llevar a cabo dicha práctica.

La Integración Continua es una práctica recomendada por muchas metodologías de software. Por ejemplo, fue recogida como una de las doce prácticas originales de la Programación Extrema[31] y forma parte de las recomendaciones del Proceso Unificado[32].

En realidad, la Integración Continua se considera un miembro más del equipo de desarrollo que se encarga de:

- Monitorizar el código fuente.
- Compilar cada cambio.

[31] XP eXtreme Programming o Programación Extrema, es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, preocupándose por el aprendizaje de los desarrolladores y propiciando el buen clima de trabajo.

[32] UP Unified Procesos o Proceso Unificado, es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental.

- Probar el resultado una vez compilado mediante pruebas automatizadas.
- Notificar a los responsables cualquier problema que se haya producido durante el proceso.

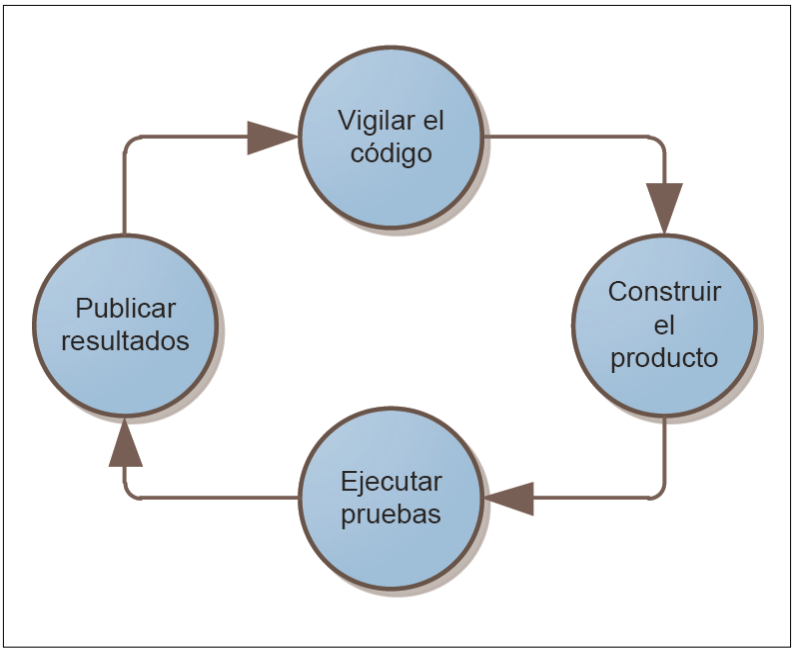


Figura 1: Ciclo de vida de la integración continua
Fuente: (GARCÍA DÍAZ, 2011)

La Fig. 1 ilustra el proceso de Integración Continua y en la Fig. 2 se detalla el mismo.

Existen dos tipos de Integración Continua, aunque a veces es difícil conocer cuál es la línea que separa una clasificación de otra. Por un lado están los centralizados, utilizados por grandes empresas como Microsoft, y por otro lado están los descentralizados, de uso mayoritario por su sencilla aplicación. La diferencia entre ambos tipos es que en la segunda opción, la decisión de cuando integrar y la responsabilidad para conseguir una integración satisfactoria es de cada desarrollador individual.

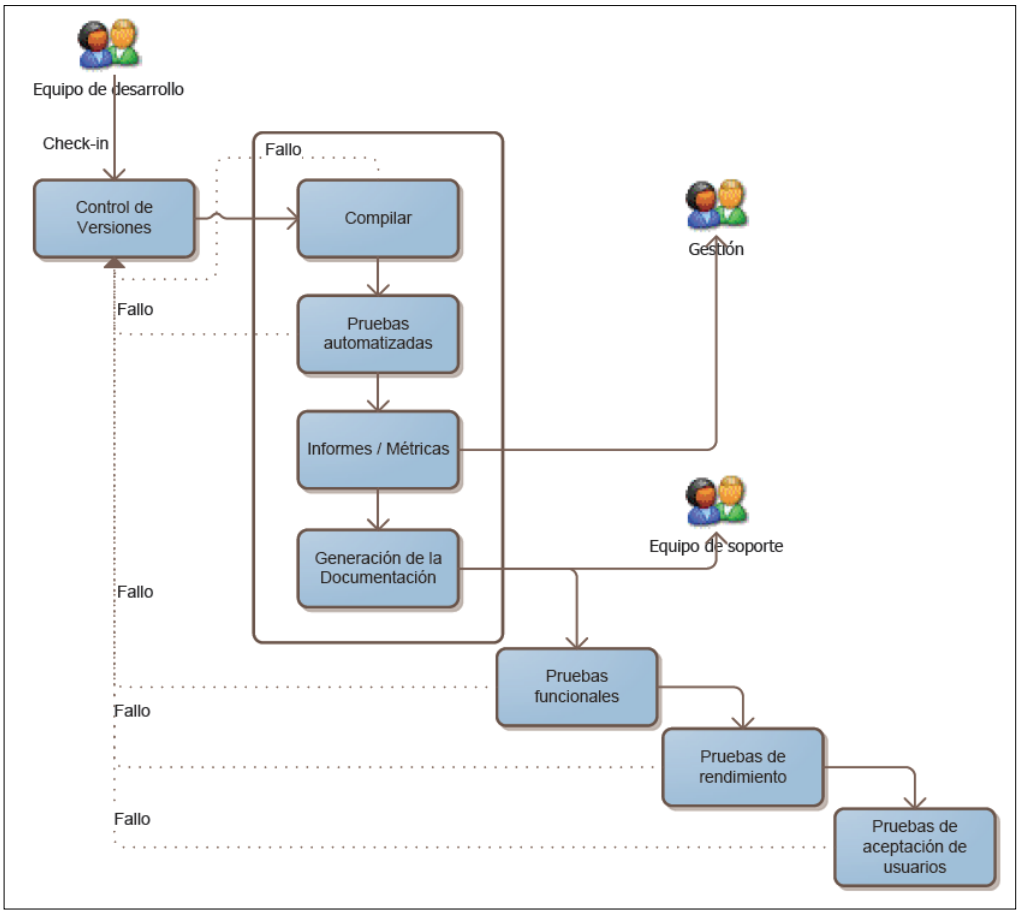


Figura 2: Ciclo de vida de la Integración Continua (extendido). Fuente: (GARCÍA DÍAZ, 2011)

Cabe destacar que existen pocos trabajos científicos relacionados con la Integración Continua; uno de los más destacados estudia cómo los desarrolladores trabajan en dos proyectos de código abierto de éxito, FreeBSD[33] y Mozilla realizado por Holck y Jorgensen[34] entre los años 2003 y 2004. La conclusión de este estudio fue que la Integración Continua reemplaza en cierto grado a la ingeniería tradicional del software respecto a mecanismos de coordinación y documentos de diseño.

[33] FreeBSD es un avanzado sistema operativo para arquitecturas x86 compatibles (como Pentium® y Athlon™), amd64 compatibles (como Opteron™, Athlon™64 EM64T), UltraSPARC®, IA-64, PC-98 y ARM. FreeBSD es un derivado de BSD, la versión de UNIX® desarrollada en la Universidad de California, Berkeley. FreeBSD es desarrollado y mantenido por un numeroso equipo de personas. El soporte para otras arquitecturas está en diferentes fases de desarrollo.

[34] Continuous Integration and Quality Assurance: A Case Study of Two Open Source Projects

2.1.1 Desarrollo de Software ágil

El desarrollo ágil de software es un término acuñado en 2001 que hace referencia al conjunto de metodologías de desarrollo basadas en un desarrollo incremental, en el que tanto los requisitos de los clientes como la solución del desarrollo evolucionan mediante la colaboración entre los diferentes equipos de las organizaciones.

Debido a que el desarrollo de software ágil va ganando aceptación con el paso del tiempo, también lo hacen sus prácticas, y precisamente una de las más importantes es la Integración Continua.

2.1.2 Manifiesto Ágil

Creado por 17 críticos convocados en marzo de 2001 por Kent Beck, autor del libro Extreme Programming Explained de 1999. La idea era debatir sobre nuevas técnicas y procesos para desarrollar software como alternativa a las metodologías tradicionales formales, debido a que son consideradas excesivamente pesadas y poco flexibles.

Como resultado del debate se tienen cuatro postulados que contienen los principios en los que se basan las metodologías alternativas, dando lugar al Manifiesto Ágil, firmado por todos los integrantes de la reunión:

"Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:

1. **A los individuos y su interacción**, por encima de los procesos y las herramientas.
2. **El software que funciona**, por encima de la documentación exhaustiva.
3. **La colaboración con el cliente**, por encima de la negociación contractual.
4. **La respuesta al cambio**, por encima del seguimiento de un plan.

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda." (BECK, 2001)

2.1.3 Principios

Los firmantes del Manifiesto Ágil redactaron doce principios o prácticas que ellos mismos utilizan:

“Seguimos estos principios:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software con valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.

- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.” (BECK, 2001)

2.1.4 Adopción en el desarrollo de software

Hay un consenso en la comunidad científica respecto a que es más útil utilizar las prácticas recomendadas por las diferentes metodologías que más se adapten a las necesidades de un equipo u organización, que utilizar metodologías completas sin tener en cuenta otras prácticas que podrían ser importantes.

Como dijo Ivar Jacobson (JACOBSON, 2007):

“En el futuro no va a haber más metodologías prescriptivas como lo fueron UP, procesos estilo CMMI o procesos ágiles estilo XP, SCRUM y otros. En cambio, lo que va a haber es una paleta de prácticas. Las prácticas van a venir primero y las metodologías van a ser meras colecciones de prácticas que las organizaciones van a escoger de la paleta.”

2.1.5 Patrones recomendados

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. En cualquier práctica del día a día es aconsejable utilizar patrones y evitar anti patrones que expertos en la materia ya han identificado y validado. A continuación se listan patrones para la realización de Integración Continua que a día de hoy están siendo utilizados con éxito:

2.1.5.1 Guardar todas las fuentes en un repositorio de código

Es aconsejable que todos los proyectos de software tengan un sistema de control de versiones que permita trabajar con el repositorio en el que se almacenan las fuentes. Se deberían almacenar en un único repositorio todas las fuentes necesarias para construir los productos, teniendo en cuenta que únicamente serán necesarias las fuentes y nada de lo que se construye a partir de las mismas. Estas fuentes incluyen scripts de instalación, archivos de configuración, esquemas de la base de datos e incluso librerías de terceros. La idea es que se puedan construir los productos en una máquina que cuente únicamente con lo mínimo indispensable. También es

aconsejable que todo el proyecto siga una misma línea base y evitar el uso de múltiples ramificaciones de versiones en el repositorio salvo en casos muy puntuales.

2.1.5.2 Automatizar la construcción

Desplegar el sistema que se construye es una tarea complicada que implica muchas operaciones como: compilaciones, movimiento de archivos y/o inserción de datos en una base de datos. Como la mayoría de dichas operaciones puede ser automatizada, la idea es que nunca se hagan de forma manual. Para automatizar la construcción del software se utilizan herramientas como Ant, NAnt o MSBuild, que analizan qué partes del proceso tienen que ser cambiadas y evitan de ese modo tener que realizar toda la construcción para cada cambio.

2.1.5.3 Utilizar pruebas automatizadas

Una buena manera de detectar errores rápida y eficientemente es incluir pruebas automáticas en el proceso de construcción, de modo que si algo falla, el software no se genera. Obviamente, las pruebas probablemente no detecten la totalidad de los errores en el código, pero detectarán una gran cantidad de ellos. En los últimos tiempos, han experimentado un gran auge aproximaciones de desarrollo de software como XP o TDD (Test Driven Development) que apuestan fuertemente por las pruebas automatizadas. Para lograrlo, se suelen utilizar herramientas como JUnit o NUnit.

2.1.5.4 Actualizar el repositorio todos los días

Cada desarrollador debería actualizar su copia con la información del repositorio cada día, realizar pruebas en su copia, y si todo es correcto, subirlo al repositorio compartido por todo el equipo de desarrollo. De este modo se evita que haya muchas diferencias entre una actualización y otra; además permite solucionar rápidamente posibles conflictos que pueden ocurrir en el código desarrollado paralelamente por diferentes personas. Así, será mucho más probable detectar un defecto que si ha pasado mucho tiempo y el software ha seguido evolucionando.

2.1.5.5 Construir siempre el software

Pese a las grandes ventajas que puede ofrecer la INTEGRACIÓN CONTINUA, es muy complicado concienciar a todos los desarrolladores que han de actualizar su

copia regularmente, después probarla y por último subir todos los cambios al repositorio. Así, muchas veces se actualizan los repositorios sin tener el software probado, provocando fallos y problemas. También, es frecuente que los desarrolladores utilicen distintas configuraciones software para trabajar, con lo que a unos les funciona, a otros les falla.

2.1.5.6 Lograr construcciones rápidas

En algunas ocasiones la construcción del software se hace un proceso demasiado lento, siendo la ejecución de ciertas pruebas como las de carga o rendimiento un cuello de botella típico, requiriendo incluso horas para ser llevadas a cabo. Para evitar este tipo de problemas se hacen construcciones por etapas. La idea es tener dos o más construcciones realizándose en secuencia, aunque sólo una será la construcción que se iniciará directamente cuando el desarrollador actualiza el repositorio. La construcción inicial deberá contener pruebas básicas que no requieran un tiempo excesivo para llevarse a cabo y será con la que trabajen el resto de desarrolladores. Sin embargo, cuando haya tiempo y recursos, se irán haciendo el resto de construcciones de la secuencia, que al ser secundarias pueden llevarse a cabo en mucho más tiempo (incluso horas).

2.1.5.7 Probar en un clon del equipo de producción

Cuantas más diferencias haya entre el equipo en el que se hacen las pruebas y el equipo en el que se desplegará el sistema final, más riesgo habrá de que se produzcan problemas. Recientemente está creciendo el uso de máquinas virtuales que son clones del sistema final. De ese modo, se consigue hacer muchas pruebas o simular muchas máquinas, en un único equipo.

2.1.5.8 Obtener la última versión ha de ser fácil

Todas las personas implicadas en el desarrollo de software han de poder obtener la última versión del software y ejecutarla fácilmente. Así, podrán ver los últimos cambios o hacer pruebas, con la seguridad de que están trabajando con la versión de software correcta.

2.1.5.9 Saber el estado de la última versión del software

Del mismo modo que todo el equipo ha de poder obtener fácilmente la última versión del software, también sería conveniente que pudieran visualizar de algún modo el estado de dicha versión. Por ejemplo, se pueden utilizar Webs, correos electrónicos o archivos de registro de eventos en los que se dan detalles de la última construcción.

2.1.5.10 Automatizar el despliegue

Es aconsejable tener automatizado el proceso de despliegue, ya que ahorra una gran cantidad de tiempo. Por ese motivo, se suelen emplear scripts para poder desplegar el software directamente. Además, es recomendable disponer de mecanismos de rollback para evitar problemas y poder volver a un estado anterior cuando se desee.

2.1.6 Ventajas de la Integración Continua

Gracias a la Integración Continua se pueden obtener unos interesantes beneficios, algunos de los cuales has sido citados en trabajos (McCONNELL, 2004). A continuación se muestra una lista de beneficios que diversos autores han identificado:

2.1.6.1 Reducción del riesgo

El riesgo es algo intrínseco a los proyectos dado su carácter temporal y único. Tiene su origen en la incertidumbre que está presente en todos los proyectos, y es por ello que la gestión de riesgos es la parte fundamental de la gestión activa de un proyecto [PMI, 2005].

Sin embargo, en el desarrollo de software todavía no se tiene una mentalidad tan global y ello provoca que no se tengan en cuenta los riesgos que hay que asumir cuando se integran diferentes componentes software de un proyecto. Generalmente, la integración de los componentes se hace durante las últimas etapas planificadas del proyecto, y con el añadido de que es poco frecuente que un proyecto llegue a la etapa de integración según las previsiones iniciales. Este hecho hace que fracasen muchos proyectos debido a que no se entregan a tiempo, se aumentan los costes o se tiene

que reducir su alcance. Con la Integración Continua se reduce el riesgo dado que los problemas durante el desarrollo se detectan antes.

2.1.6.2 Eliminación de defectos

Gracias a la Integración Continua es mucho más fácil detectar y eliminar defectos, ya que al hacerse pruebas en cortos periodos de tiempo, se aumentan las garantías de que en el software no surjan fallos, o que al menos puedan aparecer muchos menos de los que aparecerían de no ser por la integración temprana. Además, siempre es mucho más sencillo corregir defectos justo después de que aparezcan, sabiendo en qué fragmento del código pueden estar, que hacerlo mucho tiempo después. Por otra parte, los defectos son acumulativos y hay un factor psicológico que hace que resulte mucho más frustrante corregir muchos defectos al mismo tiempo, que hacerlo de uno en uno [Hunt and Thomas, 1999].

2.1.6.3 Mejores relaciones con los clientes

Realizar despliegues de manera frecuente y rápida es una gran ventaja porque puede ayudar a eliminar las barreras que existen entre los clientes y los desarrolladores. Esto permite observar las características del software rápida y fácilmente, y así crear un ambiente más colaborativo entre ambas partes.

2.1.6.4 Aumento de la moral del equipo de desarrollo

Gracias a que los desarrolladores ven rápidamente los resultados de su trabajo mediante la realización de despliegues iterativos y progresivos, se produce un aumento de su moral.

2.1.6.5 Estimaciones más acertadas

Como se tiene una idea más real del estado del desarrollo del proyecto en cada momento, se pueden hacer estimaciones de cuánto tiempo falta para acabar un proyecto con una mayor habilidad que de forma tradicional. Una de las características de la Integración Continua es que evita la aparición del llamado big bang, originado cuando se realiza la integración únicamente al final del ciclo de desarrollo, provocando la aparición de una gran cantidad de errores e invalidando las estimaciones realizadas (PRESSMAN, 2010)

2.1.6.6 Disponibilidad de la última versión del código

Gracias a la Integración Continua se puede tener acceso inmediato a la última versión de las fuentes integradas de un proyecto. De ese modo, resulta mucho más fácil hacer pruebas, demostraciones, liberación de versiones, etc.

2.1.6.7 Mejora de las capacidades colaborativas del equipo

El impacto negativo de introducir artefactos en el repositorio que no funcionan y que pueden ser incompatibles con los del resto, hace que los miembros del equipo realicen un trabajo más incremental y cuidadoso, teniendo siempre presente el trabajo de los demás.

2.1.6.8 Reducción de costos

Como se puede observar en la Figura 2-3, cuanto más se tarda en detectar un error, más costosa será su reparación. Por medio de la INTEGRACIÓN CONTINUA, los errores se detectan antes que utilizando prácticas de metodologías más tradicionales.

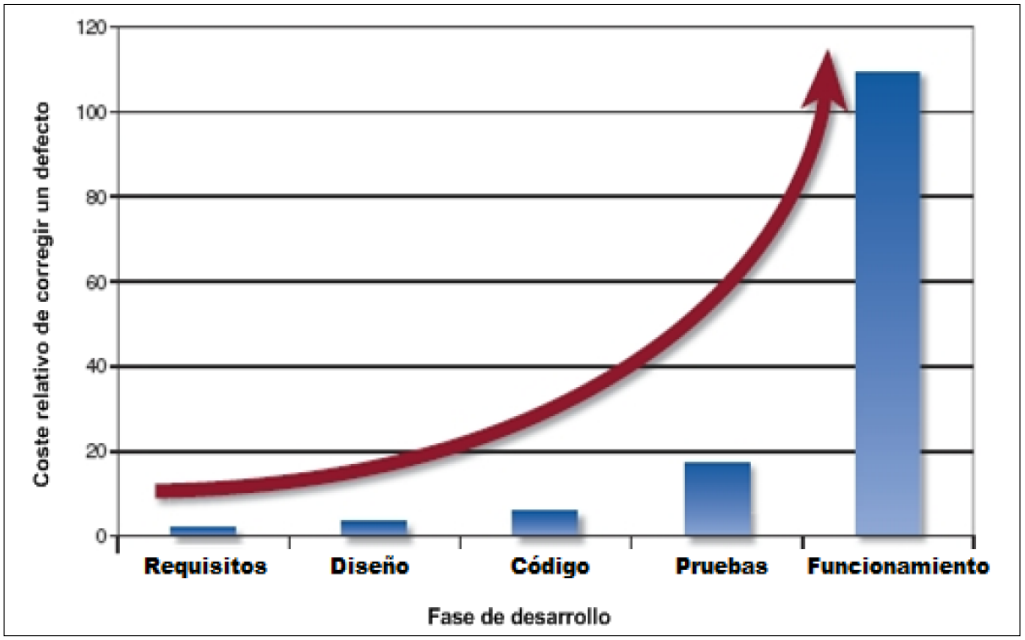


Figura 3: Costo de corregir un defecto según la fase de desarrollo.
Fuente: (GARCÍA DÍAZ, 2011).

Existen otros datos económicos que justifican el uso de la Integración Continua. Barry Boehm, uno de los mayores expertos en calidad del software, ha publicado varios estudios que demuestran cómo el coste de eliminar un defecto en un software crece exponencialmente en cada fase en la que aún no se haya descubierto dicho fallo (BOEHM, 2002). Otros trabajos han confirmado las teorías de Boehm.

En (SHARPE, 2003) se explica que reparar un defecto durante la etapa de desarrollo de un módulo podría tener un coste aproximado de un dólar. Sin embargo, la reparación costaría más de cien dólares cuando ya se ha integrado todo el código y miles de dólares si el defecto se detecta cuando el software ya se ha distribuido. Un estudio del NIST (National Institute of Standards and Technology o Instituto Nacional de Estándares y Tecnologías) ha revelado que los defectos en el software hacen que los Estados Unidos se gasten 60 billones de dólares cada año (TASSEY, 2002). También el NIST ha reconocido que casi el 80% del coste total de los proyectos se destina a corregir sus defectos.

2.1.7 Inconvenientes de la Integración Continua

Pese a las múltiples ventajas, la Integración Continua también tiene inconvenientes que detallan a continuación:

2.1.7.1 Degeneración de la arquitectura

Se ha detectado una degeneración de la arquitectura debido a que los desarrolladores se centran más en el corto plazo.

2.1.7.2 Sobrecarga del sistema

Debido a que continuamente se hacen integraciones, el sistema se verá sobrecargado, aunque no es un problema muy importante dada la potencia que actualmente tienen los ordenadores.

2.1.7.3 Necesidad de un servidor

Generalmente la integración se hace en un ordenador diferente al de los desarrolladores para que todos puedan acceder a él en cada momento. Este hecho provoca la necesidad de tener un servidor dedicado, lo cual es asumible dado el decreciente coste de los ordenadores.

2.1.7.4 Miedo a subir código erróneo

Algunos desarrolladores sientan miedo por el hecho de tener que subir continuamente código a un repositorio. Pese a todo, siempre será más problemático subir el código completo en una única iteración que hacerlo progresivamente con la práctica de la Integración Continua.

2.1.7.5 No todos utilizan correctamente los repositorios

Hay desarrolladores que cargan código con errores en los repositorios haciendo que la construcción falle. Este problema se evita concienciando a los desarrolladores de la necesidad de realizar pruebas.

2.2 Análisis Estático de Software

2.2.1 ¿Qué es el análisis estático de software?

"El análisis estático del código es el proceso de evaluar el software sin ejecutarlo"

Es, por tanto, una técnica que se aplica **directamente** sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo. La idea es que, en base a ese código fuente, podamos obtener información que nos permita mejorar la base del código **manteniendo la semántica original**.

El analizador estático de código, para ello, recibirá el código fuente de nuestro programa, lo procesará intentando averiguar qué es lo que queremos que haga y **nos dará sugerencias** con las que poder mejorar ese código.

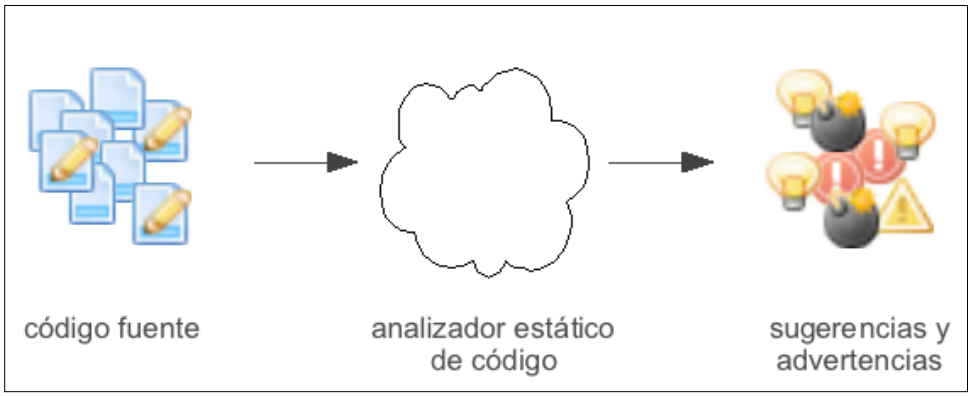


Figura 4: Esquema general del análisis de código.
Fuente: (EXPÓSITO, 2009).

Pero, ¿cómo hace esto?, ¿qué hace para "saber" qué es lo que queremos hacer y qué podemos hacer para mejorarlo? Estas herramientas incluyen, por un lado, analizadores léxicos y sintácticos que procesan el código fuente y, por otro, un conjunto de reglas que aplicar sobre determinadas estructuras. Si nuestro código fuente posee una estructura concreta que el analizador considere como "mejorable" en base a sus reglas nos lo indicará y nos sugerirá una mejora.

Y todo esto, ¿para qué sirve?, ¿qué salimos ganando realizando estos análisis? Básicamente **ganamos en facilidad de mantenimiento y de desarrollo** ya que su objetivo es **minimizar la deuda técnica**[35] de nuestros proyectos, y es que algunas de las funciones de los analizadores consisten en encontrar partes del código que puedan:

- reducir el rendimiento,
- provocar errores en el software,
- complicar el flujo de datos,
- tener una excesiva complejidad,
- suponer un problema en la seguridad.

Como podéis ver su misión es la de servirnos de ayuda en nuestros desarrollos, ayudándonos a detectar nuestros propios errores y ofreciéndonos posibles soluciones a los mismos. Nos ofrecen mucho y no nos piden nada a cambio, así que, ¿por qué no utilizarlos?

2.2.2 ¿Qué tipos de análisis estático del código existen?

Podríamos decir que existen dos. Por un lado está el análisis automático que realiza un programa de ordenador sobre nuestro código y por otro está el análisis manual que realiza una persona. Cada uno de estos análisis persigue unos objetivos concretos:

[35] "La metáfora de la técnica, desarrollada por Ward Cunningham, explica cómo el proceso de desarrollar de forma 'rápida y sucia' nos hace incurrir en una deuda, que al igual que una deuda financiera, nos obliga al pago de intereses, que se traducen en un esfuerzo extra a realizar en las siguientes iteraciones de desarrollo" (SOGETI Innovation Today, 2014)

- El análisis realizado por un programa de ordenador, o análisis automático, reduce la complejidad que supone detectar problemas en la base del código ya que busca utilizando unas reglas que tiene predefinidas.
- El análisis realizado por una persona, o análisis manual, se centra en apartados propios de nuestra aplicación en concreto como, por ejemplo, determinar si las librerías que está utilizando nuestro programa se están utilizando debidamente o si la arquitectura de nuestro software es la correcta.

Ambos, por tanto, son complementarios. El análisis automático se centra únicamente en facetas de más bajo nivel como la sintaxis y la semántica del código, funcionando este análisis en cualquier tipo de aplicación, mientras que el análisis manual se ocupa de facetas de más alto nivel como, por ejemplo, la estructura de nuestra aplicación o su manera de trabajar con otros elementos externos como las librerías.

La unión de ambos tipos de análisis nos permitirá identificar los potenciales problemas a distintos niveles que generen la deuda técnica de nuestro proyecto. Debemos, por tanto, unificar ambas para poder mejorar la base de nuestro código fuente, lo cual repercutirá en una mejora tanto del desarrollo como del mantenimiento de nuestro software antes incluso de llegar a ejecutarlo.

2.2.3 ¿Cuándo debemos hacer estos análisis?

Ahora que conocemos las ventajas del análisis estático del código, que sabemos lo útil que es, que sabemos que existe el análisis manual y el automático y que sabemos que nos va a ayudar a hacer nuestros desarrollos cabe preguntarnos ¿cada cuánto debo realizar este análisis?, ¿todos los días antes de acostarme?, ¿cada vez que escriba una nueva línea de código en mi aplicación?, ¿o una vez al año que no hace daño?

Debemos tener en mente que el análisis estático del código es un medio con el cual podemos conseguir mejorar nuestro código fuente, no un fin en sí mismo. Debemos, por tanto, usarlo únicamente como apoyo. Una diferencia entre el análisis automático y el análisis manual es el tiempo que éste tarda en realizarse. De este modo un análisis automático se realiza en pocos minutos mientras que uno manual

puede tardar varias horas. Esto, como es natural, influye a la hora de establecer una periodicidad.

Deberíamos tratar de realizar el análisis manual cada vez que vayamos a crear una nueva funcionalidad en nuestro software, preguntándonos en este caso si la arquitectura actual nos permite implementarla con facilidad, si disponemos de las librerías adecuadas o si podemos modificar la base de nuestro código para facilitar el desarrollo de esta nueva funcionalidad. Es decir, lo reservaremos para situaciones concretas.

Como es natural también podremos hacer este tipo de análisis cuando el proyecto sencillamente va mal, cuando nos cueste mucho esfuerzo desarrollar nuevas funcionalidades o modificar las que ya tenemos o, en definitiva, cuando sintamos que las piezas del software que estamos desarrollando chirrían, aunque este último caso no es el deseable y es precisamente lo que queremos evitar con el análisis estático del código.

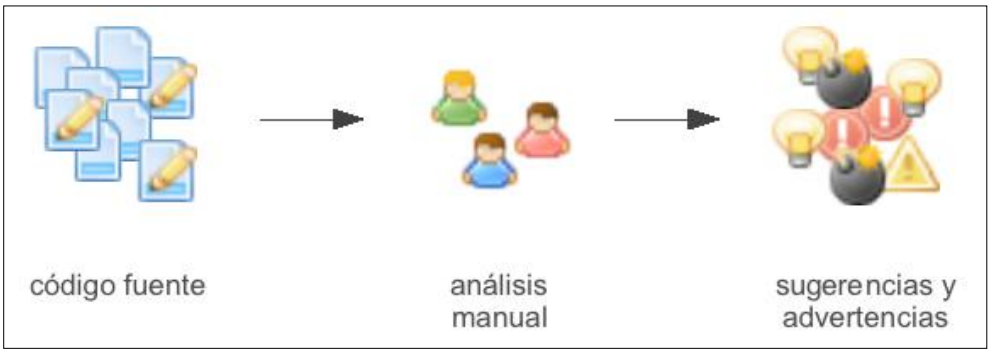


Figura 5: Análisis manual del código fuente.
Fuente: (EXPÓSITO, 2009).

El análisis automático, en cambio, puede ser realizado con una mayor periodicidad ya que no requiere de intervención humana y, lo que es mejor, puede ser programado y repetido tantas veces como sea necesario. Además obra con objetividad, siempre nos va a devolver la misma respuesta ante el mismo código fuente de entrada. Sin embargo no tenemos que creernos literalmente lo que nos diga. Puede que entre sus reglas esté determinada una condición de error concreta que, en nuestro código en particular, no debería ser etiquetada como tal.

El caso ideal sería integrar nuestro analizador automático con alguna opción de Integración Continua o alguna tecnología que permita generar la aplicación a partir de nuestro código fuente. De este modo cada vez que hagamos algún cambio o que queramos generar nuestra aplicación se ejecutará el analizador automático y este, a su vez, nos hará sugerencias.

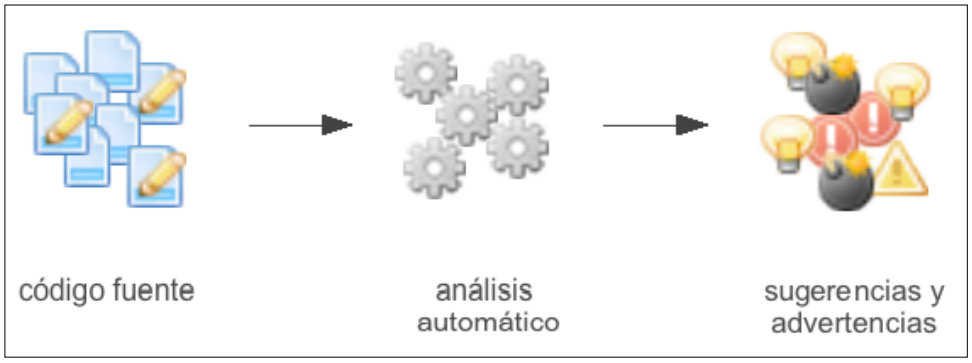


Figura 6: Análisis automático de código fuente.
Fuente: (EXPÓSITO, 2009).

Los desarrolladores, por lo general, cuando implementamos por completo alguna funcionalidad en nuestros ordenadores dejamos nuestro código fuente en un repositorio centralizado donde el resto de desarrolladores puedan encontrarlo. Un buen momento para analizar automáticamente el código es el instante en el que ese código pasa de estar únicamente en nuestros ordenadores a formar parte del repositorio, ya que así todo el equipo de desarrollo puede ser consciente de las posibles mejoras que se puedan realizar sobre ese código.

Sin embargo, es posible que queramos que el código ya se encuentre mejorado antes incluso de formar parte del repositorio. Los desarrolladores disponemos, por lo general, de herramientas que transforman nuestro código fuente en aplicaciones. Si utilizamos nuestro analizador automático en nuestro propio ordenador cuando convertimos nuestro código fuente en aplicaciones sólo nosotros recibiremos las sugerencias y podremos seguirlas antes de poner el código a disposición de todo el equipo de desarrollo. En cualquier caso estas no son reglas de oro ya que el momento en el que se deba realizar el análisis estático del código dependerá del proyecto concreto y de sus circunstancias.

2.3 Administración de Librerías y Dependencias

Cuando se inicia la construcción de un proyecto Java, lo más común es descargar desde diferentes sitios de Internet las librerías de las herramientas escogidas como base y sobre las que se desarrollará un producto determinado. Estas librerías se colocan en alguna parte del disco duro de la estación de trabajo para luego invocarlas desde el IDE[36] o incorporarlas en la configuración del entorno de desarrollo. Mientras que el equipo de trabajo está conformado por una sola persona no hay problema pero, cuando se habla de separación de trabajo en base a modularidad o capas entonces, pueden presentarse problemas como que, se escogieron diferentes versiones de la misma librería, que los entornos de desarrollo están configurados de forma diferente o que la forma de integrar las librerías al producto cambie de un programador a otro. Estos escenarios se vuelven más críticos cuando hablamos de diferentes equipos de trabajo dentro del proceso de desarrollo como es el caso del paso a control de calidad para la validación del producto previo a su paso a producción en donde, normalmente es necesario construir desde cero el mismo ambiente que logró el desarrollador para construir su sistema. Es entonces donde se puede perder un valioso tiempo tratado de incorporar las librerías de las que depende el nuevo producto para primero lograr que el código entregado compile y posteriormente para que haga lo que tiene que hacer sin que falle en el intento. Es aquí en donde hace falta un gestor de las librerías de las que dependerá el software que se esté desarrollando. El gestor de dependencias se explica más adelante en este mismo capítulo.

[36] IDE del inglés Integrated Development Environment o Entorno de Desarrollo Integrado en español, es un programa compuesto por un conjunto de herramientas para un programador, editor de código, compilador, depurador y constructor de interfaz gráfica.

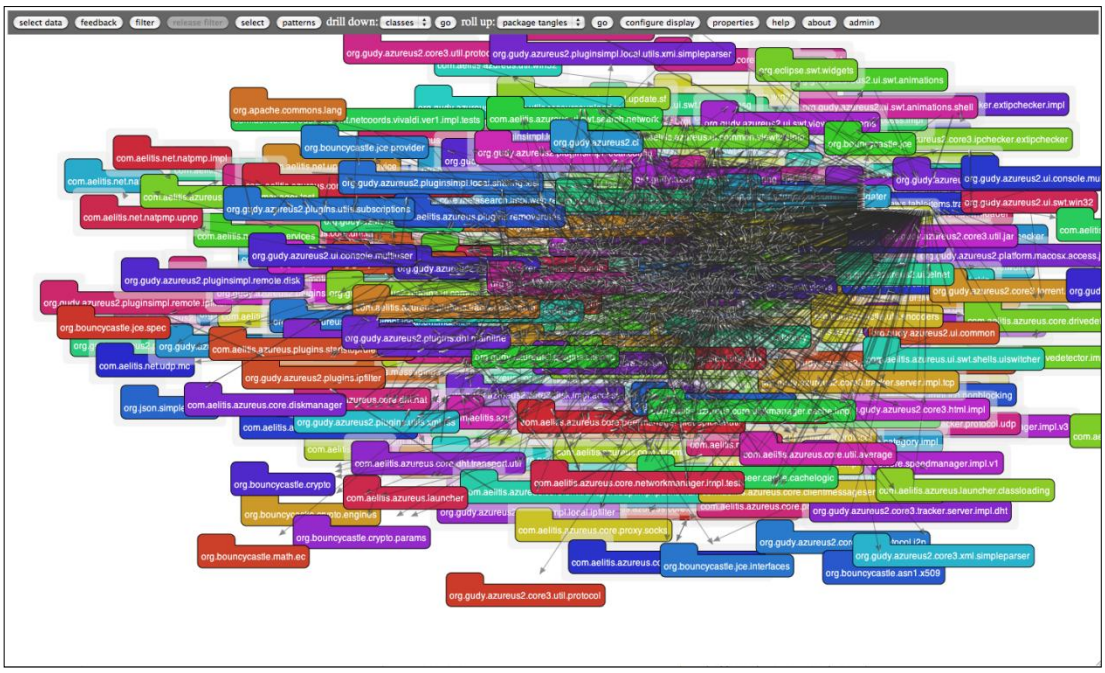


Figura 7: Diagrama de dependencia de componentes
Fuente: (DIETRICH, 2011).

En la automatización de la construcción en Java, la gestión de las dependencias se ha convertido en una parte que se constituye como un papel importante para una herramienta de automatización de la construcción, pero esto no fue siempre así. La gestión de las dependencias de un proyecto fue básicamente colocar un archivo jar en una carpeta de liberación y luego agregarlo a su sistema de control de versiones. O, para los más aventureros, se puede crear un script que descargue la biblioteca en la versión deseada de una fuente externa, y rezar para que la URL externa no haya cambiado.

Hacer todo esto de forma manual puede ser un proceso desalentador y complicado, especialmente teniendo en cuenta las dependencias transitivas (dependencias de dependencias), que sólo podrían aparecer si se ejecuta ciertas ramas de código de la librería. Esto entonces puede causar que cosas inesperadas sucedan durante el tiempo de ejecución, a pesar de que este es un problema que podría haberse arreglado en tiempo de compilación con una buena identificación y resolución de.las dependencias transitivas.

Por suerte, en la actual generación de herramientas de automatización de la construcción este escenario ya no es predominante. Con la continua popularidad que ha ganado la modularidad, la necesidad de interacción entre proyectos, así como las dependencias externas ha crecido, y como resultado todas las herramientas de automatización de construcción más utilizadas han afrontado ese reto e incluyen soporte

para la gestión de dependencias; ya sea fuera de la ellas, o por medio de plugins. Y para hacer las cosas aún más simples para el desarrollador, todas ellas utilizan una sintaxis similar para la definición de las dependencias, así como también son capaces de obtener las dependencias de los mismos repositorios públicos de artefactos (por ejemplo, Maven Central). Pero tan conveniente como son los repositorios públicos de artefactos, también introducen una nueva serie de complicaciones; conflictos entre las dependencias transitivas, y la carga añadida de buscar las librerías en un repositorio remoto.

2.4 Control de versiones del código fuente

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. El control de versiones es útil para guardar cualquier documento que cambie con frecuencia, como el código fuente de un programa.

La razón por la cual el control de versiones es universal es porque ayuda virtualmente en todos los aspectos al dirigir un proyecto: comunicación entre los desarrolladores, manejo de los lanzamientos, administración de fallos, estabilidad entre el código y los esfuerzos de desarrollo experimental y atribución y autorización en los cambios de los desarrolladores.

El control de código fuente es un tema importante durante el desarrollo de software, ya que nos permite primero que nada respaldar nuestro trabajo en un servidor central, y adicionalmente otros usuarios pueden trabajar sobre dichos archivos. En el caso de un daño grave de la computadora donde trabajamos no perderíamos algo tan valioso como el código que hemos generado.

2.5 Herramientas para la implementación de un ecosistema de Desarrollo de Software

“Un ecosistema de desarrollo software es un espacio de trabajo en el que conviven una serie de herramientas que acompañadas de unas buenas prácticas permiten a un equipo de desarrollo modelar una metodología de trabajo” (RECENA, 2008).

2.5.1 Motor de Integración Continua

En los equipos de trabajo de desarrollo de software se busca reducir los tiempos de las actividades que se realizan a través de la automatización. Esto ha llevado a que en los últimos años hayan surgido nuevas prácticas y herramientas que tienden a satisfacer esta necesidad en el marco de la mejora continua del proceso de desarrollo de software. En este sentido, la Integración Continua, de la cual devienen los motores de integración, es uno de los temas que está ocupando un lugar cada vez más importante en la construcción de software.

Un motor de Integración Continua es un proceso o procedimiento automático que toma el código fuente colocado en un repositorio centralizado para construir un producto de software y ejecutar pruebas que faciliten la detección temprana de errores. Es el mecanismo automático de construcción de aplicaciones, ejecutar pruebas y otras tareas. Del uso de este tipo de herramientas se espera:

- Detectar errores mucho antes de llegar al ambiente de producción
- Automatizar el proceso de construcción para mantener siempre una versión estable del proyecto
- Lograr un producto final robusto y constantemente verificado
- Reducir riesgo y aumentar la visibilidad del proyecto

Con esto se asegura que siempre exista una versión disponible del proyecto para su revisión normalmente Control de Calidad identificando rápidamente errores que pudieran ser subidos por los desarrolladores.

2.5.2 Servidor de Control de Versiones

Es un equipo ya sea físico o virtual en el cual se implementa un sistema de control de versiones.

2.5.2.1 ¿Qué es un sistema de control de versiones?

Es casi inconcebible hoy en día trabajar desarrollando software y no utilizar un sistema para controlar los cambios que se van realizando, tanto por un desarrollador como por sus compañeros de trabajo. Tales utilidades se denominan sistemas de

control de versiones, y existe un buen puñado de ellas. La característica principal en que se dividen bien podría ser si se trata de un sistema centralizado o no. En el primer caso, existe un servidor común donde se encuentra el código fuente, tanto la versión actual en desarrollo como todas aquellas versiones intermedias desde que dio comienzo el proyecto. En el último caso, no es necesario (aunque a menudo es recomendable) poseer un servidor común, sino que se pueden enviar y recibir actualizaciones de cada uno de los miembros participantes de forma directa.

2.5.2.2 ¿Cómo funciona un sistema de control de versiones?

Lo habitual es que cada programador realice los cambios necesarios en el código fuente para la tarea que se le ha encomendado. Una vez que dichos cambios están listos, los envía al servidor (o a los otros participantes), de modo que el resto pueda recibirlos en cualquier momento, y así trabajar sobre dichos cambios cuando tengan que realizar cualquier otra tarea. Se puede dar el caso de que varios programadores trabajen sobre el mismo fichero o ficheros, en cuyo caso el sistema lo detectará, y actuará para evitar posibles conflictos. Se pueden dar dos casos:

Los programadores han trabajado en porciones de código diferentes: En principio, no se han pisado las líneas en las que han trabajado, así que es probable que sea suficiente efectuar ambos cambios sobre el fichero, sin más. Casi todos los sistemas de control de versiones detectan esta situación y realiza la unión de los cambios de forma automática, notificándolo al usuario para que tenga constancia.

Los programadores han trabajado en líneas de código comunes, modificando, eliminando o añadiendo líneas en la misma porción de código: En estos casos, el sistema suele señalar que hay un conflicto entre ambos cambios, y habitualmente genera un fichero intermedio convenientemente marcado para que se puedan revisar ambos cambios de forma simultánea, y así quedarse con uno, con el otro, o con una combinación de los dos, realizando la unión a mano y eliminando lo que sobra.

Si bien sólo por esta característica ya merece la pena trabajar con un sistema de este tipo, espera a conocer las otras ventajas:

- Se puede volver a cualquier punto del desarrollo para ver qué aspecto tenía un determinado fichero de código, o volver a una versión donde todo funcionaba antes de haber cometido el error.

- Se puede trabajar en distintas características de forma simultánea, guardando los cambios en cada una de ellas, y uniéndolos al desarrollo principal si ya han sido lo suficientemente probadas. O sencillamente puedes crear una nueva versión para probar un experimento, o corregir un defecto que se acaba de detectar en producción, sin comprometer lo que ya llevas realizado. Estas distintas ramas de trabajo hacen que veamos el repositorio de código como un árbol, donde cada una de las ramas representan experimentos que se van creando, y que luego vuelven a unirse al tronco principal del árbol (la versión que pretende llevarse a producción).
- Se puede revisar quién realizó un determinado cambio, y cuándo lo hizo.

2.5.3 Gestor de repositorios

Normalmente en la fase de diseño de un proyecto de software es necesario establecer si se construirá todo desde cero o se utilizarán elementos preexistentes como librerías desarrolladas por otras empresas, instituciones o comunidades. En el mundo de Java existe gran cantidad de recursos desarrolladas para resolver un sin número de problemas así como, la implementación de marco de trabajo basados en patrones ya probados y de los cuales se ha confirmado su validez. Si bien esto último puede considerarse como una ventaja, también termina convirtiéndose en un problema a la hora de escoger cual es la más apropiada para nuestro proyecto. Es así que nos vemos abocados a decidir entre una de varias herramientas y las diferentes versiones de cada una de ellas y de manera intrínseca las librerías de las cuales dependen. Las prácticas de desarrollo basadas en IDEs conjuntamente con una mala documentación normalmente introducen problemas a la hora de trasladar la responsabilidades a otros miembros del equipo de trabajo y/o cuando se entrega el código para su instalación en ambientes de pruebas y producción. Se ve entonces la necesidad de buscar un mecanismo estándar de manejo de las librerías de las cuales dependerá el software que se está desarrollando y los que pudieran desarrollarse a futuro, disponer de esas librerías cuando sean necesarias para futuros desarrollo o mantenimientos, saber en donde se obtuvieron dichos productos, buscar abstraer al

desarrollado de las molestias y posibles complicaciones a la hora de acceder a las librerías y su resolución a la hora de compilar y construir (build) el software. Un gestor de repositorios es un servidor diseñado para manejar repositorios de componentes binarios (librerías). Actúa como servidor proxy dedicado para acceder a repositorios público de librerías. Provee de repositorios para desplegar productos desarrollados internamente.

2.5.4 Gestor de Calidad del Código

Es una herramienta que facilita la gestión de la calidad del código fuente basándose en múltiples métricas. Dentro del contexto del desarrollo de aplicaciones con la plataforma Java, la gestión de la calidad del código fuente se orienta a:

- Elevar la calidad de los productos de software, aplicando reglas de verificación
- Reducir defectos de programación buscando errores conocidos. Una causa para esto es la incorporación de programadores de diferentes niveles de experiencia (Junior, Senior, …)
- Emitir informes de los errores identificados
- Controlar que los cambios en el código no impacten negativamente en el producto

Para lograr todo esto se busca el apoyo de una herramienta que entre otras cosas permita comprobar la calidad del código fuente de la aplicación en base a:

- Las mejores prácticas de desarrollo
- Pruebas estáticas
- Emisión de informes de la verificación del software

CAPÍTULO 3

3 MEMORIA TÉCNICA

3.1 Análisis de la situación actual y determinación de requerimientos

El Banco Central del Ecuador, paulatinamente ha incrementado los servicios que ofrece a través de la Internet o a través de una red privada que se comparte con el Sistema Financiero Nacional. Estos servicios están soportados por un conjunto de servidores de aplicación en donde se despliegan los productos desarrollados principalmente por la Dirección de Desarrollo Informático. La infraestructura sobre la que operan los servicios a los que acceden tanto las Instituciones del Sistema Financiero Nacional, las Instituciones del Sector Público y el público en general es gestionada por la Dirección de Infraestructura y Operaciones de Tecnología de la Información.

El proceso de puesta en producción de un sistema o cualquier producto de software que trabaje sobre los servidores de aplicaciones puede describirse de la siguiente manera:

- Una vez que el usuario responsable del proyecto da su visto bueno al producto en desarrollo o en mantenimiento, el equipo de trabajo de Desarrollo Informático solicita su paso a producción.
- Con el documento de aprobación del usuario, Gestión de Calidad solicita a Infraestructura y Operaciones que se levante o prepare un ambiente de pruebas con la versión final del producto para proceder a su validación.
- Infraestructura y Operaciones prepara el ambiente de pruebas en base a la documentación entregada por Desarrollo Informático.
- Infraestructura y Operaciones genera el producto a ser validado por Gestión de Calidad.

- Infraestructura y Operaciones despliega el producto obtenido en el ambiente preparado anteriormente.
- Gestión de Calidad aplica el plan de pruebas entregado por Desarrollo Informático.
- Si las pruebas son satisfactorias, Gestión de Calidad solicita a Infraestructura y Operaciones cargar el producto en el ambiente de producción, si se encuentran novedades, solicita a Desarrollo Informático que se dé solución a las misma

3.1.1 Problemas identificados

- Limitación de tiempo en cuanto a fechas de cumplimiento.
- Falta de equipos para preparar nuevos ambientes de pruebas, debido principalmente porque la falta de estándares o la falta de su cumplimiento provoca que un nuevo producto requiera condiciones diferentes a las que se configuraron para otras aplicaciones.
- Documentación inexistente, limitada, desactualizada o con errores.
- La compilación o construcción del producto se orienta a trabajar con el IDE.
- La configuración del IDE es manual y en base a la documentación entregada por el equipo desarrollador.
- El personal encargado del despliegue no tiene porque ser experto en el manejo de los IDEs. No todo el personal que trabaja en Infraestructura y Operaciones ha pasado por una larga etapa como desarrollador así como tampoco el personal de Gestión de Calidad por lo que muchas veces cualquier problema que se presente en el trabajo con el IDE puede terminar convirtiéndose en un cuello de botella realizar sus tareas.

- La disponibilidad de los Deployers y la preparación de las estaciones de trabajo puede provocar que se extiendan los tiempos para preparar el ambiente de pruebas, la generación del producto y su despliegue.
- Actualizaciones incompletas del código. Si el código modificado no es cargado en Sistema de Control de Versiones, pueden presentarse resultados inesperados que normalmente distraen la atención del proceso porque puede asumirse que se dieron errores en la configuración o la creación del producto.
- Direcciones IP y puertos quemados en el código fuente. Es habitual que los desarrolladores se ajusten al ambiente en el que están trabajando y utilicen direcciones IP asociadas a sus estaciones de trabajo así como los puertos de los servidores locales que utilizan para las primeras pruebas del producto.
- Rotación de personal elevada debido a nuevas condiciones laborales en el Sector Público
- El desarrollo de aplicaciones web involucran un mayor grado de complejidad que las que involucran las tradicionales aplicaciones Ciente / Servidor
- El cambio de Deployer involucra la necesidad de volver a preparar el ambiente de compilación y construcción del producto sobre su estación de trabajo
- Evolución de la tecnología (nuevos frameworks, librerías, etc) promueven que no se mantenga estable o estándar la arquitectura sobre la que se desarrollan las aplicaciones
- Gestión de Calidad se orienta a aplicar pruebas de caja negra para validar que el sistema cumpla con los requerimientos del usuario
- No existen pruebas orientadas a verificar la calidad del código del sistema

- La participación de Gestión de Calidad previa la fase de Control Calidad es mínima

En el escenario descrito, puede notarse que conforme se presenten inconvenientes, el proceso de validación del nuevo software puede dilatarse. Suele suceder que para la fase de Control de Calidad se tiene un tiempo limitado debido a los plazos establecidos al inicio del proyecto y las presiones por el cumplimiento de las fechas comprometidas.

3.1.2 Requerimientos

Del proceso descrito y los problemas identificados anteriormente se desprenden los siguientes requerimientos

- Centralizar el proceso de construcción del producto de software en un ambiente independiente del Deployer de turno para mantener una configuración estándar y estable
- Automatizar tareas repetitivas como pueden ser la descarga del código fuente, la compilación, y construcción del producto.
- Incorporar y ejecutar pruebas unitarias de código (JUnit) y si es posible automatizarla.
- Ejecutar de forma automática pruebas estáticas de código.
- Identificar de forma temprana de errores o defectos y aplicar soluciones rápidas.
- Acelerar el tiempo de entrega a producción.
- Propiciar entregas continuas al cliente.
- Reducir riesgos tecnológicos buscando que el proceso de construcción sea predecible, retroalimentado y transparente.

3.1.3 Entorno de desarrollo

Cuando se inició el desarrollo de aplicaciones con Java, se adoptaron las práctica que se venían manejando en la construcción de sistemas con la arquitectura Cliente / Servidor con PowerBuilder. El código fuente se entregaba aprovechando directorios compartidos en un servidor de archivos y una vez puesta en producción se descuidaba el código que generó el producto que inició sus operaciones.

Como parte del desarrollo del Sistema del Depósito Centralizado de Valores (DCV), se adoptaron varias herramientas para apoyar el proceso de desarrollo de software en base a la recomendación hecha por la empresa contratada para construir este producto pero, sin el sustento teórico ni práctico que permitiera el aprovechamiento de estas herramientas.

Es así que se introdujo Subversion como sistema de control de código fuente, Maven como herramienta de construcción de los productos y Nexus como Repositorio de Librerías. Con estas herramientas se buscaba responder a interrogantes como:

- ¿En dónde almacenar el código fuente de las aplicaciones Java?
- ¿Cómo llevar el control de los cambios?
- ¿Cómo unificar los cambios realizados por varios programadores?
- ¿Cómo resolver inconvenientes cuando varias personas modifican los mismos archivos?
- ¿Cómo regresar a una versión operativa?
- ¿Cómo evitar el acceso no autorizado al código?
- ¿Cómo generar la estructura de directorios de los proyectos?
- ¿Cómo gestionar la compilación y el empaquetado?
- ¿Cómo gestionar las dependencias con las librerías de terceros?
- ¿Dónde almacenamos los componentes de terceros?
- ¿Dónde almacenamos los productos generados por la organización?

3.2 Análisis de las herramientas disponibles en el mercado

Según el sitio web continuousintegrationtools.com tenemos un gran número de herramientas de Integración Continua, luego de verificar su continuidad se han escogido siguientes para el análisis.

3.2.1 Comerciales

UrbanCode Build: Reúne la capacidad de respuesta y la agilidad de la automatización con seguridad basada en roles, estricta auditabilidad y trazabilidad.

Bamboo: Servidor de Integración Continua de la empresa Atlassian, creadores de JIRA, Confluence y Crowd.

ElectricFlow: ElectricFlow de Electric Cloud es una suite de productos que permiten la entrega continua mediante la automatización de los procesos de construcción, prueba y despliegue. ElectricFlow entrega a los equipos de Desarrollo y Operaciones (Dev y Ops) el control, visibilidad compartida y capacidades específicas de dominio necesarias para automatizar los procesos de construcción, prueba y despliegue a gran escala.

FinalBuilder Server: FinalBuilder Server es un poderoso servidor de construcción automática y de Integración Continua.

OpenMake Meister: OpenMake Meister proporciona un servidor de Integración Continua de nivel empresarial que permite ciclos de construcción iniciados por el proceso de actualización en un sistema de control de versiones con un completo flujo de trabajo y gestión del proceso de construcción.

OpenMake Mojo: alinea el reparto de las tareas operacionales entre los desarrolladores y los equipos de control de producción. Crea un proceso centralizado para definir los pasos necesarios para crear, probar y desplegar software.

Parabuild: Parabuild es un sistema de gestión de compilación del software y la liberación que ayuda a los equipos de software para liberar a tiempo, proporcionándoles prácticamente irrompible liberación construye y de Integración Continua.

Pulse: Pulse es un servidor de Integración Continua que es fácil de configurar y fácil de usar mientras que proporciona características avanzadas.

QuickBuild: QuickBuild es un servidor de integración y gestión de liberación continua. Proporciona un lugar central para producir, implementar el software de prueba y las versiones de lanzamiento.

Railsonfire: Railsonfire ofrece Integración Continua y el despliegue continuo de código Ruby alojado en GitHub.

TeamCity: TeamCity es un sistema de Integración Continua y gestión de construcción.

Zed: Provee el poder de la construcción y el manejo de defectos en una aplicación web integrada.

3.2.2 Código Abierto

Apache Continuum: Es un servidor de Integración Continua listo para la empresa con funciones como construcción automatizada, gestión de versiones, seguridad basada en roles, y la integración con herramientas de construcción populares y sistemas de gestión y control de código. Si usted tiene un equipo de trabajo de construcción centralizada o quiere poner el control de publicaciones en las manos de los desarrolladores, Continuum puede ayudarle a mejorar la calidad y mantener un entorno de construcción consistente.

Apache Gump: Gump es una herramienta de Integración Continua de Apache. Está escrito en Python y es totalmente compatible con Apache Ant, Maven (1.x y 2.x) y otras herramientas de construcción. Gump es único en que construye y compila software contra las últimas versiones de desarrollo de esos proyectos. Esto permite a Gump detectar cambios potencialmente incompatibles en el software sólo unas horas después de que esos cambios se registraron en el sistema de control de versiones. Las notificaciones se envían al equipo del proyecto en cuanto se detecta un cambio de este tipo, haciendo referencia a informes más detallados disponibles en línea.

BuildBot: El BuildBot es un sistema para automatizar el ciclo de compilación / prueba requerido por la mayoría de los proyectos de software para validar los cambios de código. Al reconstruir y probar el árbol cada vez que algo ha cambiado de forma automática, se identifican rápidamente los problemas de construcción, antes de que otros desarrolladores sean incomodados por el fallo. El desarrollador culpable puede ser identificado sin intervención humana. Corriendo la ejecución en una

variedad de plataformas, los desarrolladores que no tienen las instalaciones para probar sus cambios antes de actualizar el repositorio de código podrán saber rápidamente si han roto la construcción o no. Conteo de advertencia, chequeos, tamaño de la imagen, el tiempo de compilación, y otros parámetros de construcción pueden ser rastreados a través del tiempo, son más visibles, y son por lo tanto más fácil de mejorar.

Cruise Control: Cruise Control es un marco de trabajo para un proceso de construcción continua. Incluye, pero no se limita a, los plugins para la notificación de correo electrónico, Ant, y varias herramientas de control de fuentes. Una interfaz web se proporciona para ver los detalles de la actual y anterior construcción.

Cruise Control.NET: Cruise Control.NET es un servidor de Integración Continua automatizada, implementado utilizando Microsoft .NET Framework.

Cruise Control.rb: Cruise Control.rb es una herramienta de Integración Continua. Su propósito básico en la vida es alertar a los miembros de un proyecto de software cuando uno de ellos comprueba algo en control de código fuente que rompe la compilación.

easyCIS: Trae una solución fácil para el proceso de integración, construcción y gestión. Está diseñado como un sistema con una interfaz intuitiva y fácil de usar, pero no limitado en la solidez y el alcance de sus características.

Go: Automatiza y agiliza el ciclo de creación-pruebas-publicación para la entrega continua de su producto sin preocupaciones.

Jenkins / Hudson: Supervisa ejecuciones de trabajos repetitivos, como la construcción de un proyecto de software o trabajos ejecutados por demonio cron. Entre otras cosas, Jenkins / Hudson se centra en los siguientes dos trabajos:

- Construir / probar continuamente los proyectos de software, al igual que CruiseControl. En pocas palabras, Jenkins / Hudson ofrece una herramienta fácil de usar llamado sistema de Integración Continua, lo que facilita a los desarrolladores integrar cambios en el proyecto, y lo que es más fácil para los usuarios obtener una construcción nueva. Las continuas construcciones automatizadas aumentan la productividad.

- Supervisión de la Ejecuciones de trabajos dirigidos externamente, tales como trabajos de cron, incluso aquellos que se ejecutan en una máquina remota. Por ejemplo, con cron, todo lo que se recibe son correos electrónicos regulares que captan la salida, y que depende de los participantes para mirarlos con diligencia y notar cuando se rompió. Jenkins / Hudson mantiene esas salidas y hace que sea fácil darse cuenta cuando algo está mal.

Luntbuild: Luntbuild es una herramienta de automatización y gestión de construcción de gran alcance. Integración Continua o construcciones nocturnas, se puede configurar fácilmente mediante una interfaz web limpia. Las compilaciones realizadas son bien manejados usando funciones como la búsqueda o categorización, promoción, parches, eliminación, etc. También actúa como un repositorio central de artefactos construidos y área de descarga para todo el equipo de trabajo.

Sin: Es un marco de trabajo para la implementación de Integración Continua en la parte superior del sistema de control de versiones Subversion. Esto puede ayudar a mejorar la productividad general de su equipo de desarrollo al reducir el número de experiencias de rotura de construcción o incluso eliminándolas por completo.

3.2.2 Estudio o diseño de la arquitectura candidata

Partiendo de definición de Martin Fowler podemos evaluar que herramientas son necesarias:

"La Integración Continua es una práctica de desarrollo de software en la cual los miembros de un equipo integran su trabajo frecuentemente, como mínimo de forma diaria." Hasta aquí la herramienta que se identifica es, un sistema de control de versiones en donde el desarrollador o los desarrolladores colocarían el código nuevo o modificado; a continuación tenemos, "Cada integración se verifica mediante una herramienta de construcción automática para detectar los errores de integración tan pronto como sea posible", ya en este punto se identifica la necesidad del motor de

Integración Continua que sería en encargado de disparar el procedo que con el apoyo de la herramienta de construcción automática compile y empaquete la nueva versión del producto.

Adicionalmente Fowler presenta un escenario en el cual se describe un ejemplo de un nuevo desarrollo siguiendo las indicaciones de la Integración Continua:

Se inicia obteniendo una copia local del código fuente desde el repositorio o sistema de control de versiones, donde se lo guarda de forma integrada. Sobre la copia local o copia de trabajo se procede a realizar los cambios necesarios en el código y en las pruebas de verificación que se ejecutarán de forma automatizada. Las pruebas corresponden a pruebas unitarias. Con los cambios realizados se llevará a cabo una construcción automatizada en la máquina de desarrollo. Esta fase se encargará de compilar el código, empaquetarlo y realizar las pruebas automatizadas. Sólo si todas estas tareas se realizan sin error se considera que la construcción ha sido correcta. Si hay errores, habrá que solucionarlos.

Tras una construcción correcta ya se puede pensar en entregar los cambios al repositorio. Puesto que puede haber habido otros cambios por parte otros desarrolladores mientras se trabaja en local, lo primero será actualizar la copia local y reconstruir el proyecto. En caso de conflicto con los cambios propios aparecerán errores en la compilación o en las pruebas que se tendrán que solucionar antes de poder actualizar el repositorio con los cambios.

Una vez entregados los se volverá a construir la línea principal de desarrollo del proyecto en la máquina de integración. Si no hay errores se ya podrá decir que los cambios se han llevado a cabo. En caso de que apareciesen errores, habría que solucionarlos. Esta construcción automática en la máquina de integración puede ser realizada de forma manual o de forma automática mediante alguna herramienta.

En caso de que aparezcan conflictos entre desarrolladores, estos serán detectados rápidamente. En este momento lo más importante será solucionar estos errores tan pronto como sea posible. El resultado de todo esto es un proyecto que funciona correctamente y tiene pocos errores. Todo el mundo desarrolla a partir de un código estable y trata de mantenerse lo más cerca de él como para que las integraciones con él no lleven demasiado tiempo. Se tarda menos tiempo en arreglar errores porque estos aparecen más rápidamente.

En este ejemplo podemos identificar nuevamente el repositorio de código del cual se obtiene la copia de trabajo, un IDE con el cual los desarrolladores pueden realizar los cambios en el código y en las pruebas de verificación y una herramienta de construcción automática que operaría en el ambiente de integración. Todo esto bajo la orquestación del motor de Integración Continua.

Hasta aquí se han identificado las herramientas que permitan establecer una arquitectura básica en donde, el motor de Integración Continua es una solución administrable gráficamente sobre motores de desarrollo y plataformas que proporcionan una centralización e integración del código de desarrollo, con la posibilidad de hacer un seguimiento en todo momento y capacidad de establecer métricas de calidad, tanto en el código individual como en la integración del trabajo del equipo.

Para centralizar el código se usa el servidor de repositorio de código o Sistema de Control de Versiones, el motor de integración puede monitorizar ese repositorio y cuando existan cambios obtiene una copia del mismo, ejecuta tareas de integración y verifica que se cumpla con las métricas de calidad. La figura 3.1 muestra un esquema general de cómo estaría dispuesta, y la interacción de las herramientas en un entorno de Integración Continua.

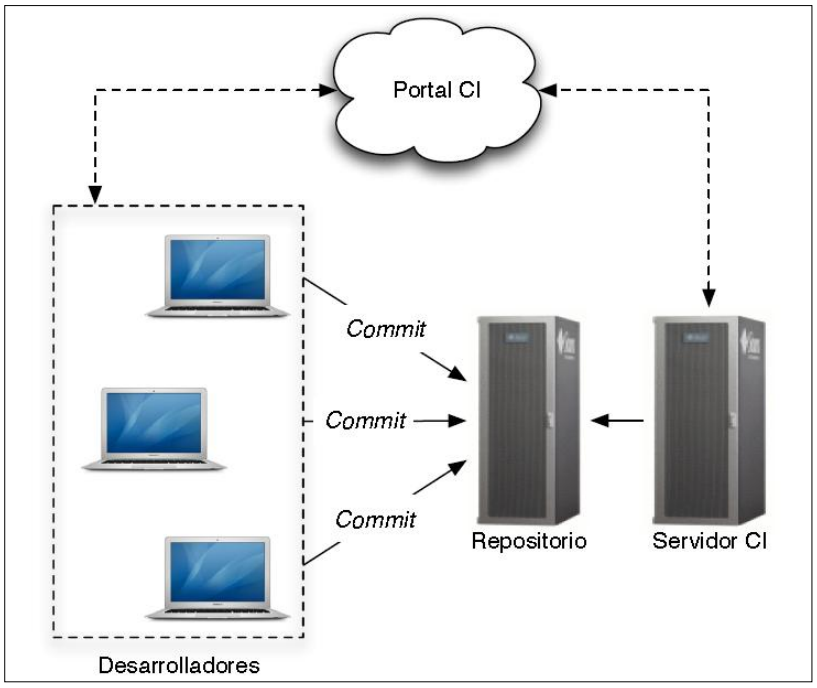


Figura 1: Arquitectura básica de Integración Continua.
Fuente: (CAÑADILLAS MEDINA, 2010)

Una vez identificados los principales elementos de la Infraestructura de Integración Continua, se procederá a identificar las alternativas que mejor se adapten al ámbito de trabajo previsto para su uso.

3.3 Selección de las herramientas

Se ha tomado de referencia la "Guía Técnica sobre Evaluación de Software en la Administración Pública" elaborada por la Oficina de Gobierno Electrónico e Informática de la Presidencia del Consejo de Ministros del Gobierno del Perú la cual a su vez, está basada en la norma ISO/IEC 9126 de la ISO[37] y la IEC[38] que forman el sistema especializado para la normalización internacional.

Esta guía plantea un "Proceso de Evaluación de Software" y propone un conjunto de pasos listados en la sección metodología del capítulo 1 de este trabajo, estos pasos son desarrollados a continuación.

3.4.1 Propósito de la evaluación

Analizar la herramienta que mejor se adapte al entorno operativo en el que se lleva a cabo el proceso de desarrollo de software en la Coordinación General de Tecnologías de la Información y Comunicación del Banco Central del Ecuado y su adecuación al modelo informático de Integración Continua.

3.4.2 Tipo de producto

La herramienta considerada es un servidor de Integración Continua conforme se ha venido describiendo a lo largo de este trabajo.

3.4.3 Modelo de Calidad

Para definir el modelo de calidad se han establecido categorías para las cualidades de la calidad del software, basadas en seis características:

- Funcionalidad
- Confiabilidad

[37] Organización Internacional de Normalización del inglés International Organization for Standardization (http://www.iso.org/).

[38] Comisión Electrónica Internacional o en inglés International Electrotechnical Commision (http://www.iec.ch/)

- Utilidad
- Eficiencia
- Capacidad de mantenimiento
- Portabilidad

En la figura 3.2 se muestra una subdivisión de las características ya mencionadas.

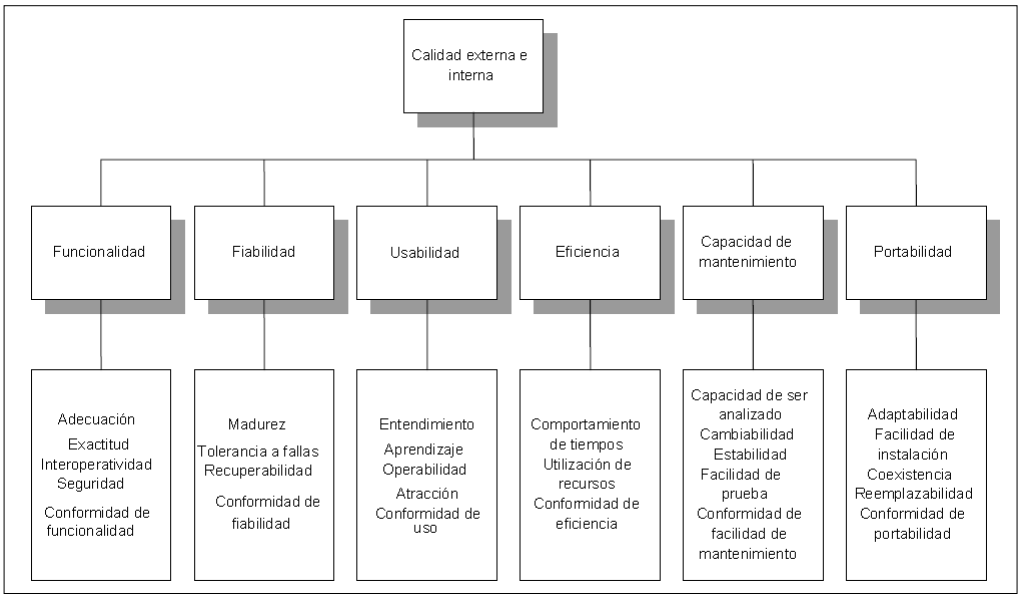


Figura 8: Subdivisión de las características de calidad del software. Fuente: (ONGEI, 2004)

Las definiciones se dan para cada característica y sub característica de calidad del software que influye en la calidad. Para cada característica y sub característica, la capacidad del software es determinada por un conjunto de atributos internos que pueden ser medidos. Las características y sub características se pueden medir externamente por la capacidad provista por el sistema que contiene el software.

3.4.3.1 Funcionalidad

La capacidad del producto de software para proveer las funciones que satisfacen las necesidades explícitas e implícitas cuando el software se utiliza bajo condiciones específicas.

Esta característica se refiere a lo que hace el software para satisfacer necesidades, mientras que las otras características se refieren principalmente a cuándo y a cómo satisfacen las necesidades.

Para un sistema que es operado por un usuario, la combinación de la funcionalidad, fiabilidad, usabilidad y eficiencia puede ser medida externamente por su calidad en uso.

- **Adecuación**

La capacidad del producto de software para proveer un adecuado conjunto de funciones para las tareas y objetivos especificados por el usuario.

Ejemplos de adecuación son la composición orientada a tareas de funciones a partir de sub funciones que las constituyen, y las capacidades de las tablas.

- **Exactitud**

La capacidad del producto de software para proveer los resultados o efectos acordados con un grado necesario de precisión.

- **Interoperabilidad**

La capacidad del producto de software de interactuar con uno o más sistemas especificados. La interoperabilidad se utiliza en lugar de compatibilidad para evitar una posible ambigüedad con la reemplazabilidad.

- **Seguridad**

La capacidad del producto de software para proteger la información y los datos de modo que las personas o los sistemas no autorizados no puedan leerlos o modificarlos, y a las personas o sistemas autorizados no se les niegue el acceso a ellos.

La seguridad en un sentido amplio se define como característica de la calidad en uso, pues no se relaciona con el software solamente, sino con todo un sistema.

- **Conformidad de la funcionalidad**

La capacidad del producto de software de adherirse a los estándares, convenciones o regulaciones legales y prescripciones similares referentes a la funcionalidad.

3.4.3.2 Fiabilidad

La capacidad del producto de software para mantener un nivel específico de funcionamiento cuando se está utilizando bajo condiciones especificadas.

El desgaste o envejecimiento no ocurre en el software. Las limitaciones en fiabilidad son debido a fallas en los requerimientos, diseño, e implementación. Las fallas debido a estos errores dependen de la manera en que se utiliza el producto de software y de las opciones del programa seleccionadas, más que del tiempo transcurrido.

La definición de fiabilidad en la ISO/IEC 2382-14:1997 es "la habilidad de la unidad funcional de realizar una función requerida...". En este documento, la funcionalidad es solamente una de las características de la calidad del software. Por lo tanto, la definición de la fiabilidad se ha ampliado a "mantener un nivel especificado del funcionamiento..." en vez de "...realizar una función requerida".

- **Madurez**

La capacidad del producto de software para evitar fallas como resultado de errores en el software.

- **Tolerancia a errores**

La capacidad del producto de software para mantener un nivel especificado de funcionamiento en caso de errores del software o de incumplimiento de su interfaz especificada. El nivel especificado de funcionamiento puede incluir la falta de capacidad de seguridad.

- **Recuperabilidad**

La capacidad del producto de software para restablecer un nivel especificado de funcionamiento y recuperar los datos afectados directamente en el caso de una falla.

Después de una falla, un producto de software a veces estará no disponible por cierto período del tiempo, intervalo en el cual se evaluará su recuperabilidad.

La disponibilidad es la capacidad del producto de software para poder realizar una función requerida en un punto dado en el tiempo, bajo condiciones indicadas de uso. En extremo, la disponibilidad se puede determinar por la proporción de tiempo total, durante la cual, el producto de software está en un estado ascendente. La disponibilidad, por lo tanto, es una combinación de madurez (con control de frecuencias de fallas), de la tolerancia de errores y de la recuperabilidad (que gobierna el intervalo de tiempo en cada falla). Por esta razón es que no ha sido incluida como una sub característica separada.

- **Conformidad de la fiabilidad**

La capacidad del producto de software para adherirse a las normas, convenciones o regulaciones relativas a la fiabilidad.

3.4.3.3 Usabilidad

La capacidad del producto de software de ser entendido, aprendido, usado y atractivo al usuario, cuando es utilizado bajo las condiciones especificadas. Algunos aspectos de funcionalidad, fiabilidad y eficiencia también afectarán la usabilidad, pero para los propósitos de la ISO/IEC 9126 ellos no son clasificados como usabilidad.

Los usuarios pueden ser operadores, usuarios finales y usuarios indirectos que están bajo la influencia o dependencia del uso del software. La usabilidad debe dirigirse a todo los diferentes ambientes de usuarios que el software puede afectar, o estar relacionado con la preparación del uso y evaluación de los resultados.

- **Entendimiento**

La capacidad del producto de software para permitir al usuario entender si el software es adecuado, y cómo puede ser utilizado para las tareas y las condiciones particulares de la aplicación.

Esto dependerá de la documentación y de las impresiones iniciales dadas por el software.

- **Aprendizaje**

La capacidad del producto de software para permitir al usuario aprender su aplicación. Un aspecto importante a considerar aquí es la documentación del software.

- **Operatividad**

La capacidad del producto de software para permitir al usuario operarlo y controlarlo.

Los aspectos de propiedad, de cambio, de adaptabilidad y de instalación pueden afectar la operatividad.

La operatividad corresponde a la controlabilidad, a la tolerancia a errores y a la conformidad con las expectativas del usuario.

Para un sistema que es operado por un usuario, la combinación de la funcionalidad, confiabilidad, usabilidad y eficacia puede ser una medida considerada por la calidad en uso.

- **Atracción**

La capacidad del producto de software de ser atractivo al usuario.

Esto se refiere a las cualidades del software para hacer el software más atractivo al usuario, tal como el uso del color y la naturaleza del diseño gráfico.

- **Conformidad de uso**

La capacidad del producto de software para adherirse a los estándares, convenciones, guías de estilo o regulaciones relacionadas a su usabilidad.

3.4.3.4 Eficiencia

La capacidad del producto de software para proveer un desempeño adecuado, de acuerdo a la cantidad de recursos utilizados y bajo las condiciones planteadas.

Los recursos pueden incluir otros productos de software, la configuración de hardware y software del sistema, y materiales (Ej: Papel de impresión o diskettes).

Para un sistema operado por usuarios, la combinación de funcionalidad, fiabilidad, usabilidad y eficiencia pueden ser medidas externamente por medio de la calidad en uso.

- **Comportamiento de tiempos**

La capacidad del producto de software para proveer tiempos adecuados de respuesta y procesamiento, y ratios de rendimiento cuando realiza su función bajo las condiciones establecidas.

- **Utilización de recursos**

La capacidad del producto de software para utilizar cantidades y tipos adecuados de recursos cuando este funciona bajo las condiciones establecidas.Los recursos humanos están incluidos dentro del concepto de productividad.

- **Conformidad de eficiencia**

La capacidad del producto de software para adherirse a estándares o convenciones relacionados a la eficiencia.

3.4.3.5 Capacidad de mantenimiento

Capacidad del producto de software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, y especificaciones de requerimientos funcionales.

- **Capacidad de ser analizado**

La capacidad del producto de software para atenerse a diagnósticos de deficiencias o causas de fallas en el software o la identificación de las partes a ser modificadas.

- **Cambiabilidad**

La capacidad del software para permitir que una determinada modificación sea implementada.

Implementación incluye codificación, diseño y documentación de cambios.

Si el software va a ser modificado por el usuario final, la cambiabilidad podría afectar la operatividad.

- **Estabilidad**

La capacidad del producto de software para evitar efectos inesperados debido a modificaciones del software.

- **Facilidad de prueba**

La capacidad del software para permitir que las modificaciones sean validadas.

- **Conformidad de facilidad de mantenimiento**

La capacidad del software para adherirse a estándares o convenciones relativas a la facilidad de mantenimiento.

3.4.3.6 Portabilidad

La capacidad del software para ser trasladado de un entorno a otro. El entorno puede incluir entornos organizacionales, de hardware o de software.

- **Adaptabilidad**

La capacidad del producto de software para ser adaptado a diferentes entornos especificados sin aplicar acciones o medios diferentes de los previstos para el propósito del software considerado.

Adaptabilidad incluye la escalabilidad de capacidad interna (Ejemplo: Campos en pantalla, tablas, volúmenes de transacciones, formatos de reporte, etc.).

Si el software va a ser adaptado por el usuario final, la adaptabilidad corresponde a la conveniencia de la individualización, y podría afectar la operabilidad.

- **Facilidad de instalación**

La capacidad del producto de software para ser instalado en un ambiente especificado.

Si el software va a ser instalado por el usuario final, puede afectar la propiedad y operatividad resultantes.

- **Coexistencia**

La capacidad del producto de software para coexistir con otros productos de software independientes dentro de un mismo entorno, compartiendo recursos comunes.

- **Reemplazabilidad**

La capacidad del producto de software para ser utilizado en lugar de otro producto de software, para el mismo propósito y en el mismo entorno. Por ejemplo, la reemplazabilidad de una nueva versión de un producto de software es importante para el usuario cuando dicho producto de software es actualizado (actualizaciones, upgrades).

Reemplazabilidad se utiliza en lugar de compatibilidad de manera que se evitan posibles ambigüedades con la interoperabilidad. La reemplazabilidad puede incluir atributos de ambos, inestabilidad y adaptabilidad. El concepto ha sido introducido como una sub característica por sí misma, dada su importancia.

- **Conformidad de portabilidad**

La capacidad del software para adherirse a estándares o convenciones relacionados a la portabilidad.

3.4.3.7 Matriz de priorización de Holmes

Para establecer el modelo de calidad vamos a utilizar la matriz de priorización de Holmes para seleccionar un conjunto de factores de calidad, que proporcione una base para la especificación de requisitos de calidad y para la evaluación de la calidad de las herramientas de desarrollo que puedan incorporarse al modelo de Integración Continua.

La Matriz de Holmes nos permite categorizar o priorizar cualquier cantidad de opciones realizando una comparación entre ellas. Es una herramienta que nos permite priorizar parámetros que tienen características similares, esta matriz, permite comparar entre si los parámetros y clasificarlos en orden de importancia, por lo que puede ser utilizada para discriminar los parámetros de evaluación que se usará para seleccionar el motor de Integración Continua.

Los pasos son los siguientes:

- Elaborar una lista de parámetros a usar en la selección
- Graficar el modelo de la matriz de Holmes

- Completar los factores y la matriz. Este es el paso más importante porque se compara la importancia relativa de un factor respecto de los otros según mejor criterio. Se recomienda comparar los factores de manera horizontal.

La manera de calificación es muy sencilla:

1 Cuando el factor evaluado relativamente es más importante que su contraparte.

0 Cuando el factor evaluado es menos importante que su contraparte.

0.5 Cuando los dos factores son igualmente importantes.

Al finalizar se deben sumar los totales y otorgar un orden relativo, se realiza la suma de manera horizontal.

Tabla 2:
Matriz de Prioridad de Holmes.

PARAMETROS DE EVALUACIÓN	Funcionalidad	Adecuación	Exactitud	Interoperabilidad	Seguridad	Conformidad de la funcionalidad	Fiabilidad	Madurez	Tolerancia a errores	Recuperabilidad	Conformidad de la fiabilidad	Usabilidad	Entendimiento	Aprendizaje	Operabilidad	Atracción	Conformidad de uso	Eficiencia	Comportamiento de tiempos	Utilización de recursos	Conformidad de eficiencia	Capacidad de mantenimiento	Capacidad de ser analizado	Cambiabilidad	Estabilidad	Facilidad de prueba	Conformidad de facilidad de [illegible]	Portabilidad	Adaptabilidad	Facilidad de instalación	Coexistencia	Reemplazabilidad	Conformidad de portabilidad	SUMA
Funcionalidad	0,5																																	
Adecuación		0,5	1	0	0	1		1	0	1	1		1	1	1	1	1		1	1	1		1	1	1	1	1		1	1	1	1	1	23,5
Exactitud		0	0,5	1	0	1		0	1	1	1		1	1	0	1	1		1	1	1		1	1	1	1	1		1	1	1	1	1	22,5
Interoperabilidad		1	0	0,5	1	1		0	0	1	1		0	1	0	1	0		1	1	1		1	1	1	1	1		0	0	0	1	1	17,5
Seguridad		1	1	0	0,5	0		0	0	0	0		0	0	0	0	0		0	0	0		1	1	0	0	0		0	0	0	0	0	4,5
Conformidad de la funcionalidad		0	0	0	1	0,5		1	0	0	0		1	1	0	1	0		1	1	1		1	1	1	1	1		1	0	0	1	1	16,5
Fiabilidad							0,5																											
Madurez		0	1	1	1	0		0,5	1	1	1		1	0	0	1	1		1	1	0		1	1	1	1	1		1	0	1	1	0	19,5
Tolerancia a errores		1	0	1	1	1		0	0,5	0	0		0	0	0	1	0		0	1	0		1	1	1	1	1		0	0	0	0	0	11,5
Recuperabilidad		0	0	0	1	1		0	1	0,5	1		0	0	0	1	0		0	0	0		1	1	0	1	0		0	0	0	0	0	8,5
Conformidad de la fiabilidad		0	0	0	1	1		0	1	0	0,5		1	1	0	1	1		0	0	0		1	1	1	1	1		1	1	1	1	1	17,5
Usabilidad												0,5																						
Entendimiento		0	0	1	1	0		0	1	1	0		0,5	0	0	1	1		0	0	0		1	1	1	1	1		1	0	0	1	0	13,5

PARAMETROS DE EVALUACIÓN	Funcionalidad	Adecuación	Exactitud	Interoperabilidad	Seguridad	Conformidad de la funcionalidad	Fiabilidad	Madurez	Tolerancia a errores	Recuperabilidad	Conformidad de la fiabilidad	Usabilidad	Entendimiento	Aprendizaje	Operabilidad	Atracción	Conformidad de uso	Eficiencia	Comportamiento de tiempos	Utilización de recursos	Conformidad de eficiencia	Capacidad de mantenimiento	Capacidad de ser analizado	Cambiabilidad	Estabilidad	Facilidad de prueba	Conformidad de facilidad de [illegible]	Portabilidad	Adaptabilidad	Facilidad de instalación	Coexistencia	Reemplazabilidad	Conformidad de portabilidad	SUMA
Aprendizaje		0	0	0	1	0		1	1	1	0		1	0,5	0	1	1		0	0	1		1	1	1	1	1		1	0	1	1	1	17,5
Operabilidad		0	1	1	1	1		1	1	1	1		1	1	0,5	1	1		1	1	1		1	1	1	1	1		1	0	1	1	1	24,5
Atracción		0	0	0	1	0		0	0	0	0		0	0	0	0,5	0		0	0	0		1	1	1	1	1		0	0	0	0	0	6,5
Conformidad de uso		0	0	0	1	1		0	1	1	0		0	0	0	1	0,5		1	1	0		1	1	1	1	1		1	1	1	1	1	17,5
Eficiencia																		0,5																
Comportamiento de tiempos		0	0	0	1	0		0	1	1	1		1	1	0	1	0		0,5	1	1		0	0	0	0	0		1	1	1	1	1	14,5
Utilización de recursos		0	0	0	1	0		0	0	1	1		1	1	0	1	0		0	0,5	1		1	1	1	1	1		1	1	1	1	1	17,5
Conformidad de eficiencia		0	0	0	1	0		1	1	1	1		1	0	0	1	1		0	0	0,5		1	1	1	1	1		1	1	1	1	0	17,5
Capacidad de mantenimiento																						0,5												
Capacidad de ser analizado		0	0	0	0	0		0	0	0	0		0	0	0	0	0		1	0	0		0,5	1	0	0	0		0	0	0	0	0	2,5
Cambiabilidad		0	0	0	0	0		0	0	0	0		0	0	0	0	0		1	0	0		0	0,5	1	1	1		0	0	0	0	0	4,5
Estabilidad		0	0	0	1	0		0	0	1	0		0	0	0	0	0		1	0	0		1	0	0,5	1	1		0	1	0	0	0	7,5
Facilidad de prueba		0	0	0	1	0		0	0	0	0		0	0	0	0	0		1	0	0		1	0	0	0,5	0		0	0	0	0	0	3,5

PARAMETROS DE EVALUACIÓN	Funcionalidad	Adecuación	Exactitud	Interoperabilidad	Seguridad	Conformidad de la funcionalidad	Fiabilidad	Madurez	Tolerancia a errores	Recuperabilidad	Conformidad de la fiabilidad	Usabilidad	Entendimiento	Aprendizaje	Operabilidad	Atracción	Conformidad de uso	Eficiencia	Comportamiento de tiempos	Utilización de recursos	Conformidad de eficiencia	Capacidad de mantenimiento	Capacidad de ser analizado	Cambiabilidad	Estabilidad	Facilidad de prueba	Conformidad de facilidad de [illegible]	Portabilidad	Adaptabilidad	Facilidad de instalación	Coexistencia	Reemplazabilidad	Conformidad de portabilidad	SUMA
Conformidad de facilidad de mantenimiento		0	0	0	1	0		0	0	1	0		0	0	0	0	0		1	0	0		1	0	0	1	0,5		0	0	0	0	0	5,5
Portabilidad																												0,5						
Adaptabilidad		0	0	1	1	0		0	1	1	0		0	0	0	1	0		0	0	0		1	1	1	1	1		0,5	0	0	1	1	12,5
Facilidad de instalación		0	0	1	1	1		1	1	1	0		1	1	1	1	0		0	0	0		1	1	0	1	1		1	0,5	1	1	0	17,5
Coexistencia		0	0	1	1	1		0	1	1	0		1	0	0	1	0		0	0	0		1	1	1	1	1		1	0	0,5	0	1	14,5
Reemplazabilidad		0	0	0	1	0		0	1	1	0		0	0	0	1	0		0	0	0		1	1	1	1	1		0	0	1	0,5	1	11,5
Conformidad de portabilidad		0	0	0	1	0		1	1	1	0		1	0	0	1	0		0	0	1		1	1	1	1	1		0	1	0	0	0,5	13,5

3.4.3.8 Diagrama de Pareto

- **Principio de Pareto**

Afirma que en todo grupo de elementos o factores que contribuyen a un mismo efecto, unos pocos son responsables de la mayor parte de dicho efecto.

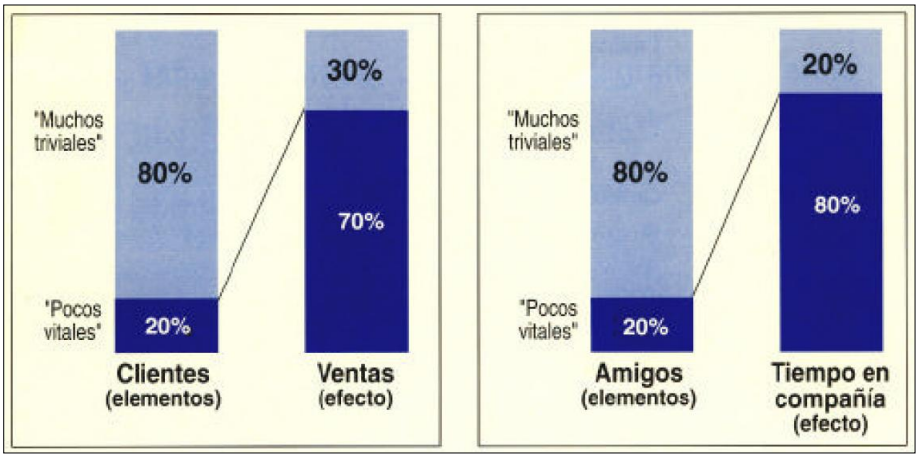


Figura 9: Principio de Pareto.
Fuente: (FUNDIBEQ, 2007) – http://www.fundibeq.com

- **Análisis de Pareto**

Definición: Es una comparación cuantitativa y ordenada de elementos o factores según su contribución a un determinado efecto.

El objetivo de esta comparación es clasificar dichos elementos o factores en dos categorías: Las "Pocas Vitales" los elementos muy importantes en su contribución y los "Muchos Triviales" los elementos poco importantes en ella.

Características principales

Priorización: Identifica los elementos que más peso o importancia tienen dentro de un grupo.

Unificación de Criterios: Enfoca y dirige el esfuerzo de los componentes del grupo de trabajo hacia un objetivo prioritario común.

Carácter objetivo: Su utilización fuerza al grupo de trabajo a tomar decisiones basadas en datos y hechos objetivos y no en ideas subjetivas.

- **Tablas y Diagramas de Pareto**

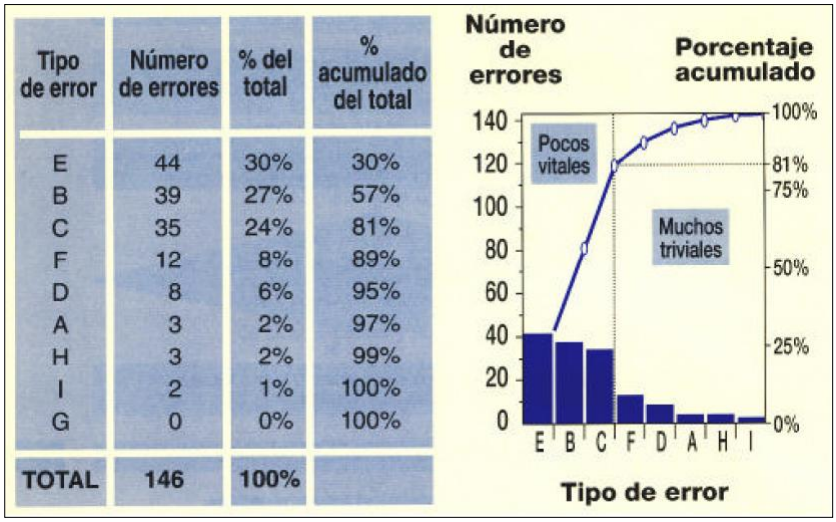
Son herramientas de representación utilizados para visualizar el Análisis de Pareto.

El Diagrama de Pareto es la representación gráfica de la Tabla de Pareto correspondiente.

Características principales

Simplicidad: Tanto la tabla como el diagrama no requieren ni cálculos complejos ni técnicas sofisticadas de representación gráfica.

Impacto visual: El Diagrama de Pareto comunica de forma clara, evidente y de un vistazo, el resultado de la comparación y priorización.



Tipo de error	Número de errores	% del total	% acumulado del total
E	44	30%	30%
B	39	27%	57%
C	35	24%	81%
F	12	8%	89%
D	8	6%	95%
A	3	2%	97%
H	3	2%	99%
I	2	1%	100%
G	0	0%	100%
TOTAL	146	100%	

Figura 10: Tabla de Pareto y Diagrama de Pareto.
Fuente: (FUNDIBEQ, 2007) – http://www.fundibeq.com

- **Proceso**

Diagrama de Flujo

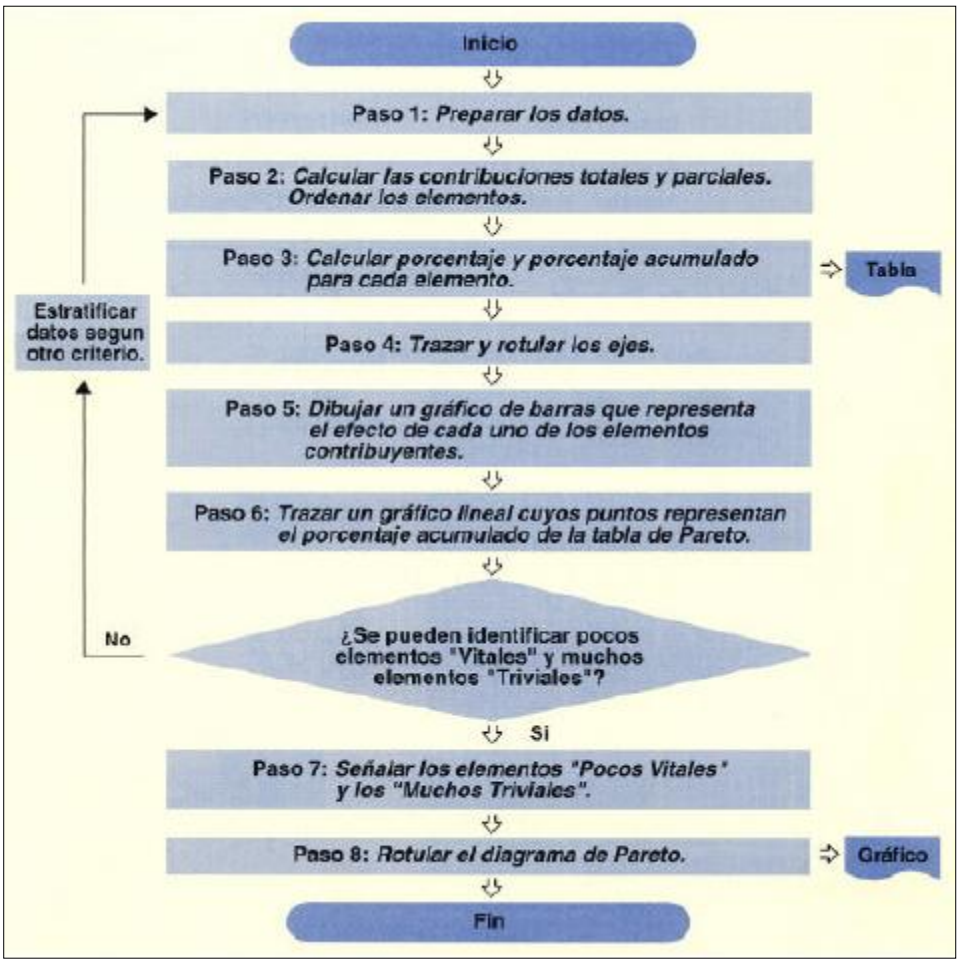


Figura 11: Diagrama de flujo para realizar un Diagrama de Pareto. Fuente: (FUNDIBEQ, 2007) – http://www.fundibeq.com

Construcción

Paso 1: Preparación de los datos

Como en todas las herramientas de análisis de datos, el primer paso consiste en recoger los datos correctos o asegurarse de que los existentes lo son.

Para la construcción de un Diagrama de Pareto son necesarios:

a) Un efecto cuantificado y medible sobre el que se quiere priorizar (Costes, tiempo, número de errores o defectos, porcentaje de clientes, etc).

b) Una lista completa de elementos o factores que contribuyen a dicho efecto (tipos de fallos o errores, pasos de un proceso, tipos de problemas, productos, servicios, etc).

Es importante identificar todos los posibles elementos de contribución al efecto antes de empezar la recogida de datos. Esta condición evitará que, al final del análisis, la categoría "Varios" resulte ser una de las incluidas en los "Pocos Vitales". Las herramientas de calidad más útiles para obtener esta lista son: la Tormenta de Ideas, el Diagrama de Flujo, el Diagrama de Causa-Efecto y sus similares, o los propios datos.

c) La magnitud de la contribución de cada elemento o factor al efecto total.

Estos datos, bien existan o bien haya que recogerlos, deberán ser:

- Objetivos: basados en hechos, no en opiniones.
- Consistentes: debe utilizarse la misma medida para todos los elementos contribuyentes y los mismos supuestos y cálculos a lo largo del estudio, ya que el Análisis de Pareto es un análisis de comparación.
- Representativos: deben reflejar toda la variedad de hechos que se producen en la realidad.
- Verosímiles: evitar cálculos o suposiciones controvertidas, ya que buscamos un soporte para la toma de decisiones, si no se cree en los datos, no apoyarán las decisiones.

Paso 2: Cálculo de las contribuciones parciales y totales. Ordenación de los elementos o factores incluidos en el análisis

Para cada elemento contribuyente sobre el efecto, anotar su magnitud. Ordenar dichos elementos de mayor a menor, según la magnitud de su contribución. Calcular la magnitud total del efecto como suma de las magnitudes parciales de cada uno de los elementos contribuyentes.

Paso 3: Calcular el porcentaje y el porcentaje acumulado, para cada elemento de la lista ordenada

El porcentaje de la contribución de cada elemento se calcula:

% = (magnitud de la contribución /magnitud del efecto total) x 100

El porcentaje acumulado para cada elemento de la lista ordenada se calcula:

- Por suma de contribuciones de cada uno de los elementos anteriores en la tabla, más el elemento en cuestión como magnitud de la contribución, y aplicando la fórmula anterior.
- Por suma de porcentajes de contribución de cada uno de los elementos anteriores más el porcentaje del elemento en cuestión. En este caso habrá que tener en cuenta el que estos porcentajes, en general, han sido redondeados.

Una vez completado este paso tenemos construida la Tabla de Pareto.

Paso 4: Trazar y rotular los ejes del Diagrama

El eje vertical izquierdo representa la magnitud del efecto estudiado.
Debe empezar en 0 e ir hasta el valor del efecto total.
Rotularlo con el efecto, la unidad de medida y la escala.
La escala debe ser consistente, es decir variar según intervalos constantes.
Las escalas de gráficos que se compararán entre sí, deben ser idénticas (Nota: Prestar especial cuidado a las escalas automáticas de los gráficos por ordenador).
El eje horizontal contiene los distintos elementos o factores que contribuyen al efecto.
Dividirlo en tantas partes como factores existan y rotular su identificación de izquierda a derecha según el orden establecido en la Tabla de Pareto.
El eje vertical derecho representa la magnitud de los porcentajes acumulados del efecto estudiado.
La escala de este eje va desde el 0 hasta el 100%. El cero coincidirá con el origen y el 100% estará alineado con el punto, del eje vertical izquierdo, que representa la magnitud total del efecto

Paso 5: Dibujar un Gráfico de Barras que representa el efecto de cada uno de los elementos contribuyentes

La altura de cada barra es igual a la contribución de cada elemento tanto medida en magnitud por medio del eje vertical izquierdo, como en porcentaje por medio del eje vertical derecho.

Paso 6: Trazar un gráfico lineal cuyos puntos representan el porcentaje acumulado de la Tabla de Pareto

Marcar los puntos del gráfico en la intersección de la prolongación del límite derecho de cada barra con la magnitud del porcentaje acumulado correspondiente al elemento representado en dicha barra.

Paso 7: Señalar los elementos "Pocos Vitales" y los "Muchos Triviales"

Trazar una línea vertical que separa el Diagrama en dos partes y sirve para visualizar la frontera entre los "Pocos Vitales" y los "Muchos Triviales", basándonos en el cambio de inclinación entre los segmentos lineales correspondientes a cada elemento.

Rotular las dos secciones del Diagrama.

Rotular el porcentaje acumulado del efecto correspondiente al último elemento incluido en la sección "Pocos Vitales".

Paso 8: Rotular el título del Diagrama de Pareto.

En base al proceso descrito en la sección anterior, se ha obtenido como resultado la Tabla y el Diagrama de Pareto, presentados en la tabla 3 en la figura 13.

Tabla 3:
Tabla de Pareto.

PARAMETROS DE EVALUACIÓN	RELEVANCIA	RELEVANCIA ACUMULADA	PORCENTAJE	PORCENTAJE ACUMULADO
Operabilidad	24,5	23,5	6,7	6,7
Adecuación	23,5	47,0	6,5	13,2
Exactitud	22,5	69,5	6,2	19,4
Madurez	19,5	89,0	5,4	24,8
Interoperabilidad	17,5	106,5	4,8	29,6
Conformidad de la fiabilidad	17,5	124,0	4,8	34,4
Aprendizaje	17,5	141,5	4,8	39,2
Conformidad de uso	17,5	159,0	4,8	44,0
Utilización de recursos	17,5	176,5	4,8	48,8
Conformidad de eficiencia	17,5	194,0	4,8	53,6
Facilidad de instalación	17,5	211,5	4,8	58,5
Conformidad de la funcionalidad	16,5	228,0	4,5	63,0
Comportamiento de tiempos	14,5	242,5	4,0	67,0
Coexistencia	14,5	257,0	4,0	71,0
Entendimiento	13,5	270,5	3,7	74,7
Conformidad de portabilidad	13,5	284,0	3,7	78,4
Adaptabilidad	12,5	296,5	3,4	81,8
Tolerancia a errores	11,5	308,0	3,2	85,0
Reemplazabilidad	11,5	319,5	3,2	88,2
Recuperabilidad	8,5	328,0	2,3	90,5
Estabilidad	7,5	335,5	2,1	92,6
Atracción	6,5	342,0	1,8	94,4
Conformidad de facilidad de mantenimiento	5,5	347,5	1,5	95,9
Seguridad	4,5	352,0	1,2	97,1
Cambiabilidad	4,5	356,5	1,2	98,3
Facilidad de prueba	3,5	360,0	1,0	99,3
Capacidad de ser analizado	2,5	362,5	0,7	100,0
TOTAL	363,5			

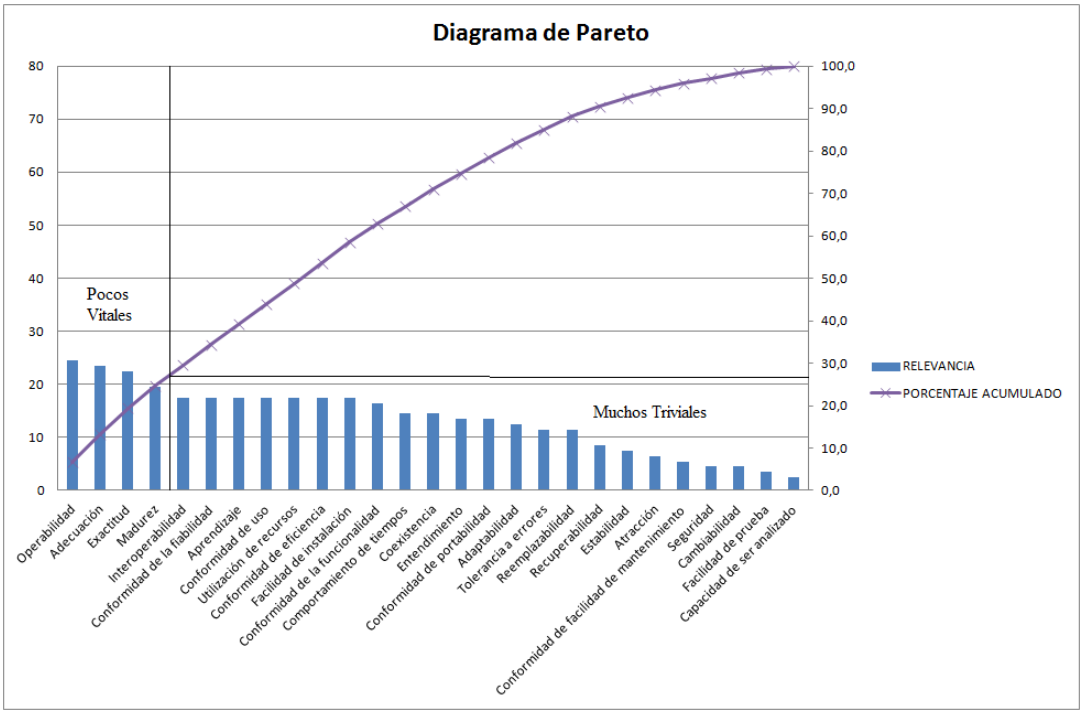


Figura 12: Diagrama de Pareto.

Del análisis realizado se desprende que cuatro de los parámetros son los más relevantes para seleccionar un servidor de Integración Continua. A estos cuatro parámetros aplicaremos nuevamente la matriz de Holmes para establecer ponderaciones que luego serán tomadas en cuenta a la hora de evaluar los productos.

PARAMETROS DE EVALUACIÓN	Operabilidad	Adecuación	Exactitud	Madurez	SUMA	PONDERACIÓN
Operabilidad	0,5	0	1	0	1,5	0,19
Adecuación	1	0,5	1	1	3,5	0,44
Exactitud	0	0	0,5	0	0,5	0,06
Madurez	1	0	1	0,5	2,5	0,31
TOTAL					**8**	**1**

Figura 13: Matriz de prioridades para los parámetros seleccionados.

Si bien se han determinado las características más representativas a ser consideradas en el modelo de calidad, estas no dejan de ser elementos de alto nivel por lo que se ha visto la necesidad de introducir atributos de las herramientas a ser evaluadas como una tercera capa de valoración en donde se establece un escala de 4 niveles para aplicarlos a dichos atributos. Los posibles valores que pueden tomar estos atributos son mostrados en la tabla 4:

Tabla 4:
Valores atributos de software.

Posibles valores de los atributos de los productos de software		
	3	Existe
	2	Existe pero es defectuosa o no bien probada
	1	Esta planeada para un futuro cercano
	0	No existe

A fin sustentar los parámetros identificados con el análisis de Holmes y el principio de Pareto, se ha procedido a relacionar un conjunto de atributos de acuerdo al tipo de producto. La figura 15 muestra esta relación.

	PARÁMETROS			
ATRIBUTOS	Operabilidad	Adecuación	Exactitud	Madurez
Soporte Gestión de Código Fuente		X		
Soporte Relacionado Gestión de Código Fuente				X
Gestión de la Contrucción	X			
Seguridad				X
Publicación	X			
Interface Web	X			
Herramientas de Construcción Directamente Soportadas	X			
Integración con Gestores de Problemas e Incidentes		X		
Inegración con Herramientas de Prueba				X
Integración con IDEs		X		
Integración Inspección de Código			X	
API Administración Remota				
Instalación y Configuración	X			

Figura 15: Relación atributos parámetros.

El detalle de los atributos considerados y su valoración es desarrollado en el siguiente capítulo.

CAPÍTULO 4

4 INFORME DE RESULTADOS

4.1 Tabulación de Resultados

Para la valoración particular del tipo de producto que se está analizando se ha considerado necesario la evaluación de atributos más puntuales y ajustados a las características que se buscan en este tipo de herramientas así, se ha considerado:

- Soporte de gestión de código fuente
- Soporte relacionado de código fuente
- Gestión de la construcción
- Seguridad
- Publicación
- Interface web
- Herramientas de construcción directamente soportadas
- Integración con gestores de problemas e incidentes
- Integración con herramientas de prueba
- Integración con IDE
- Integración inspección de código
- API de administración remota
- Instalación y configuración

Estos atributos se describen a continuación:

4.1.1 Soporte de gestión de código fuente

Capacidad de integrarse con uno o más sistemas de control de versiones. Los SCV definidos para tomarse en cuenta son:

- AccuRev
- AlienBrain
- BitKeeper
- ClearCase
- CA Harvest
- CM Synergy
- CVS
- Dimensions
- File system SCM
- Git
- archivo HTTP
- Mercurial
- MKS
- Perforce (p4)
- PVCS
- SourceGear
- Vault
- StarTeam
- Subversion
- Surround
- Team Foundation Server
- VSS
- VSS Journal

4.1.2 Soporte relacionado de código fuente

Operaciones que se pueden ejecutar en relación con un Sistema de Control de Versiones las consideradas son:

- Filtrado del SCV,
- Acceso a múltiples SCV,
- Creación de nuevos repositorios en el SCV,
- Huellas o rastro de los binarios construidos y regresados al SCV de los cuales se gestiona las versiones de código,

- Auditoria del origen de archivos utilizados en los binarios pero, no almacenados en un repositorio SCV

4.1.3 Gestión de la construcción

Facilidad de la herramienta para ejecutar las operaciones contempladas en la construcción del producto de software como:

- Construcciones paralelas (habilidad para construir muchos proyectos de forma simultánea)
- Construcciones Distribuidas
- Ejecuta compiladores y enlazadores con Build Avoidance (construcciones incrementales)
- Acelera llamadas a compiladores y enlazadores a través de procesamiento multi-hilos
- Agentes de auto actualización de código
- Realiza exploración de dependencias del código fuente, análisis de impacto y presentación de informes
- Auto generación del archivo build.xml basado en archivos de proyecto del IDE
- Construcciones forzadas manualmente
- Gestión compilado de X-platform y X-language
- Construcciones disparadas por el SCV
- Construcciones basadas en encuestas al SCV
- Construcción con calendarios
- Promoción de Construcciones

- Dependencias entre proyectos
- Borrado de construcciones
- Reproduce historial de construcciones
- Proactivo (puede prevenir roturas de las construcciones)
- Detecta nuevas pruebas fallidas mientras se construye
- Notifica cuando la primera prueba falla en la construcción

4.1.4 Seguridad

Características de control de acceso (autorización) a la herramienta y control de perfiles (autorización) apoyándose en servicios de directorio y sistemas de seguridad de la plataforma.

- Autenticación de usuario
- Esquemas de autorización de usuario
- Integración LDAP
- Kerberos
- Single Sing On
- JAAS Personalizado

4.1.5 Publicación

Notificación del resultado de la ejecución de una tarea a través de diferentes mecanismos que pueden ser:

- Confluence
- Correo electrónico (Email)
- Corre ejecutable
- FTP
- IRC
- Jabber
- ProjectStart
- RSS
- SCP
- Windows System Tray
- Formatted Logging
- Yahoo Messenger

- Lotus Sametime
- MSN Messenger
- NetSend
- X10

4.1.6 Interface Web

Actividades que se pueden realizar a través de la interface web de la herramienta, a saber:

- Ver proyectos
- Añadir nuevos proyectos
- Clonar proyectos
- Borrar Proyectos
- Modificar proyectos
- Matar construcciones
- Pausar construcciones
- Acceso a construir artefactos
- Explorar la copia de trabajo de IC
- Eliminar una copia de trabajo de IC
- Búsqueda en construcciones
- Gráficos históricos
- Página web con actualización automática
- Soporte multiproyectos
- Vista multiproyectos
- Añadir o remover agentes de equipo (para construcciones distribuidas)

4.1.7 Herramientas de Construcción Directamente Soportadas

Integración con herramientas de construcción de software sin necesidad de instalar elementos adicionales

- Script de línea de comandos o shell
- Ant
- Make
- MSBuild
- NAnt

- Groovy
- OpenMake Meister
- Maven
- Maven2
- Rake (Ruby)
- Visual Studio
- FinalBuilder

4.1.8 Integración con Gestores de Problemas e Incidentes

Capacidad de generar incidentes para ser registrados en herramientas o gestores de problemas o incidentes, la lista es la siguiente:

- Bugzilla
- ClearQuest
- Confluence
- JIRA
- Mingle
- QualityCenter
- Rally
- Rubyforge.org
- Scarab
- Sourceforge.net
- Trac
- VersionOne

4.1.9 Integración con Herramientas de Prueba

Las herramientas consideradas para la integración con el servidor de Integración Continua son:

- Agitar
- CppUnit
- JUnit
- NUnit
- QualityCenter
- PMD
- Clover
- Selenium
- SilkCentral
- MSTest

- PHPUnit

4.1.10 Integración con IDEs

Capacidad de trabajar con Entornos de Desarrollo Integrado o IDEs.

- Eclipse Plugin
- Netbeans Plugin
- VS 2005 Plugin
- IntelliJ Plugin

4.1.11 Integración Inspección de Código

Acceso al repositorio de código a través de otras herramientas como:

- Bonsai
- Fisheye
- ViewVC

4.1.12 API Administración Remota

Interfaz de programación de aplicaciones dirigida a permitir la administración remota.

- REST
- JMX
- SOAP
- Hessian
- XML-RPC
- Biblioteca de cliente

4.1.13 Instalación y Configuración

Características o facilidades de las herramientas en la fase de instalación y configuración:

- Instalador Windows
- Distribución auto contenida (excepto clientes SCV)

- Dependencias adicionales
- Plataforma de ejecución
- Plataforma de proyectos (lo que se puede construir)
- Herramienta de construcción preferida
- No requiere modificar los scripts de construcción
- Soporta múltiples proyectos
- Configuración automática desde script de construcción
- Configuración Archivo Texto

Estos atributos sustentarán la evaluación de las herramientas de Integración Continua con los parámetros o características seleccionados en el capítulo 3. Las figuras de la 16 a la 47 muestras los valores asignados a los productos evaluados y se incluye un resumen de cada valoración tanto para los productos comerciales como para los productos de uso libre.

4.2 Tabulación de resultados de herramientas comerciales

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Soporte Gestión de Código Fuente	AccuRev	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	ℹ 1	✓ 3	✗ 0
	AlienBrain	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	BitKeeper	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	ClearCase	✓ 3	✓ 3	✗ 0	✓ 3	ℹ 1	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3
	CA Harvest	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	CM Synergy	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	CVS	✓ 3	✓ 3	✗ 0	ℹ 1	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Dimensions	✓ 3	✗ 0	✗ 0	ℹ 1	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	"File system SCM"	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0
	Git	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0
	HTTP file	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	Mercurial	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	0	✗ 0	✓ 3	✗ 0
	MKS	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	Perforce (p4)	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	PVCS	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	SourceGear Vault	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	StarTeam	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3
	Subversion	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Surround	ℹ 1	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	Team Foundation Server	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
	VSS	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0
	VSS Journal	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
TOTAL		**52**	**21**	**15**	**32**	**34**	**48**	**48**	**39**	**12**	**22**	**39**	**15**

Figura 16: Valoración productos comerciales frente al Soporte de Gestión de Código Fuente.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Gestión de la Construcción	Construcciones paralelas (abilidad para construir muchos proyectos de forma simultanea)	3	3	3	3	3	3	3	3	3	3	3	3
	Construcciones Distribuidas	3	3	3	3	0	3	3	3	3	3	3	3
	Ejecuta compiladores y enlazadores con Build Avoidance (construcciones incrementales)	3	0	3	3	0	3	0	3	0	0	0	0
	Acelera llamadas a compiladores y enlazadores a través de procesamiento multi-hilos	0	0	3	3	0	3	0	0	0	0	0	0
	Agentes de auto actualización de código	0	3	0	0	0	3	3	0	3	0	3	3
	Realiza exploración de dependencias del código fuente, análisis de impacto y presentación de informes	0	0	0	3	0	3	0	0	0	0	0	0
	Auto generación del archivo build.xml basado en archivos de proyecto del IDE	0	0		0	0	3	0	0	0	0	3	0
	Construcciones forzadas manualmente	3	3	3	3	3	3	3	3	3	3	3	3

Figura 17: Valoración productos comerciales frente al Soporte Relacionado de Gestión de la Construcción (parte 1).

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Gestión de la Construcción	Gestión compilado de X-platform y X-language	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Construcciones disparadas por el SCV	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	ℹ	✓ 3	ℹ	✓ 3	✗ 0
	Construcciones basadas en encuentas al SCV	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1
	Construcción con calendarios	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Promoción de Construcciones	✓ 3	✓ 3	✓ 3	✓	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Dependencias entre proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	0
	Borrado de construcciones	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Reproduce historial de construcciones	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	ℹ 1	✓ 3	✓ 3	ℹ 1
	Proactivo (puede prevenit roturas de las construcciones)	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0
	Detecta nuevas pruebas fallidas mientras se construye	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	Notifica cuando la primera prueba falla en la construcción	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3
TOTAL		**39**	**36**	**39**	**48**	**21**	**54**	**39**	**36**	**34**	**27**	**45**	**26**

Figura 18: Valoración productos comerciales frente al Soporte Relacionado de Gestión de la Construcción (parte 2).

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Soporte Relacionado Gestión de Código Fuente	Filtrado SCV	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0
	Múltiples SCV	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0
	Puede crear nuevos repositorios en el SCV	ℹ 1	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Huellas de los binarios construidos de regreso a SCV gestionados versiones de código	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Audita archivos de origen utilizados en los binarios, pero no almacenados en un repositorio SCV	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
TOTAL		**7**	**6**	**6**	**3**	**9**	**15**	**9**	**6**	**3**	**3**	**6**	**0**

Figura 20: Valoración productos comerciales frente al Soporte Relacionado de Gestión de Código Fuente.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Seguridad	Autenticación de Usuario	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Esquemas de autorización de usuario	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Integración LDAP	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	0	✓ 3	✗ 0
	Kerberos	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Single Sign On	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	JAAS Personalizado	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	ℹ 1
TOTAL		**18**	**15**	**9**	**12**	**9**	**12**	**12**	**9**	**9**	**6**	**12**	**7**

Figura 19: Valoración productos comerciales frente a la Seguridad.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Publicación	Confluence	0	3	0	0	0	0	0	0	0	0	3	0
	Email	3	3	3	3	3	3	3	3	3	3	3	3
	Corre ejecutable	3	3	3	3	3	3	3	3	3	3	0	0
	FTP	0	3	3	3	3	3	3	3	0	0	0	0
	IRC	0	0	3	0	0	0	0	0	0	0	0	0
	Jabber	3	3	3	0	0	0	0	3	3	3	3	0
	Lotus Sametime	3	0	0	0	0	0	0	0	0	0	0	0
	NetSend	0	0	3	0	3	0	0	0	0	0	0	0
	ProjectStart	0	0	0	0	0	0	0	0	0	0	0	0
	RSS	3	3	3	3	3	0	0	3	3	0	3	3
	SCP	3	3	3	3	0	3	3	3	0	0	0	0
	Windows System Tray	3	3	3	0	3	0	0	3	3	0	3	0
	Formatted Logging	3	3	3	3	3	3	3	3	0	0	3	3
	Yahoo Messenger	3	0	3	0	0	0	0	0	0	0	0	0
	MSN Messenger	3	0	3	0	3	3	3	1	0	3	0	0
	X10	0	0	3	0	0	0	0	0	0	0	0	0
TOTAL		**30**	**27**	**39**	**18**	**24**	**18**	**18**	**25**	**15**	**12**	**18**	**9**

Figura 21: Valoración productos comerciales frente a la Publicación.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Interface Web	Ver cambios	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	Añadir nuevos proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Clonar proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Borrar proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Modificar proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Matar construcciones	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1
	Pausar construcciones	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	ℹ 1
	Acceso a construir artefactos	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Explorar la copia de trabajo de IC	✗ 0	✓ 3	✗ 0	n/a	✗ 0	✗ 0	✗ 0	ℹ 1	✓ 3	✗ 0	✗ 0	✗ 0
	Elliminar una copia de trabajo de IC	✓ 3	✓ 3	✗ 0	n/a	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
	Búsqueda en construcciones	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0
	Gráficos históricos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	ℹ 1
	Página web con actualización automática	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1
	Soporte multi proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Vista multi project	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Añadir o remover agentes de equipo (para construcciones distribuidas)	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
TOTAL		**45**	**42**	**39**	**36**	**28**	**6**	**6**	**46**	**45**	**33**	**45**	**31**

Figura 22: Valoración productos comerciales frente a la Interface Web.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Herramientas de Construcción Directamente Soportadas	Script de línea de comandos o shell	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Ant	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	Via línea de comando
	Groovy	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓	✗ 0	✗ 0	✗ 0	Via línea de comando
	OpenMake Meister	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	Via línea de comando	✗ 0
	Maven	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	Via línea de comando
	Maven2	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1	✓ 3	Via línea de comando
	Make	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	Via línea de comando	Via línea de comando
	MsBuild	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	Via línea de comando
	NAnt	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1	✓ 3	Via línea de comando
	Rake (Ruby)	ℹ 1	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	Via línea de comando
	Visual Studio ('devenv')	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	Via línea de comando
	FinalBuilder	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3
TOTAL		**28**	**36**	**36**	**30**	**18**	**27**	**27**	**30**	**24**	**11**	**18**	**6**

Figura 23: Valoración productos comerciales frente a las Herramientas de Construcción Directamente Soportadas.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Integración con Herramientas de Prueba	Agitar	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	CppUnit result rendering	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0
	JUnit result rendering	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	NUnit result rendering	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1	✓ 3	✗ 0
	QualityCenter test rendering	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	PHPUnit result rendering	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	PMD result rendering	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	Clover result rendering	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	Selenium result rendering	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0
	SilkCentral	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	MSTest result rendering	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
TOTAL		**30**	**18**	**6**	**18**	**3**	**12**	**12**	**18**	**9**	**4**	**21**	**0**

Figura 24: Valoración productos comerciales frente a Integración con Herramientas de Prueba.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Integración con Gestores de Problemas e Incidentes	Bugzilla	✓ 3	✗ 0	✗ 0	ℹ 1	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	ℹ 1	✓ 3	✗ 0
	ClearQuest	✓ 3	✗ 0	✗ 0	ℹ 1	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Confluence	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	JIRA	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	ℹ 1	✓ 3	✗ 0
	Mingle	✗ 0	✗ 0	✗ 0	✗	✗ 0	✗ 0	✗ 0	✗ 0	✗	✗ 0	✗ 0	✗ 0
	QualityCenter	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Rally	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Rubyforge.org	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Scarab	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Sourceforge.net	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Trac	ℹ 1	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3
	VersionOne	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
TOTAL		**19**	**6**	**0**	**14**	**0**	**9**	**9**	**9**	**9**	**2**	**9**	**3**

Figura 26: Valoración productos comerciales frente a Integración con Gestores de Problemas e Incidentes.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Integración Inspección de Código	Bonsai	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3
	ViewVC	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
	Fisheye	ℹ 1	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
TOTAL		**4**	**6**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**0**	**6**	**9**

Figura 25: Valoración productos comerciales frente a Integración Inspección de Código.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
API Administración Remota	REST	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
	SOAP	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	XML-RPC	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
	JMX	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Hessian	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	ℹ 1	✗ 0	✗ 0
	Biblioteca de cliente	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
TOTAL		**9**	**6**	**9**	**6**	**3**	**9**	**9**	**9**	**3**	**1**	**9**	**3**

Figura 28: Valoración productos comerciales frente a API Administración Remota.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Integración con IDEs	Eclipse Plugin	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	ℹ 1	✗ 0	ℹ 1	✓ 3	✗ 0
	VS 2005 Plugin	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
	Netbeans	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	IntelliJ Plugin	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	ℹ	✗ 0	ℹ 1	✓ 3	✗ 0
TOTAL		**6**	**6**	**0**	**6**	**0**	**6**	**6**	**1**	**0**	**2**	**9**	**0**

Figura 27: Valoración productos comerciales frente a Integración con IDEs.

ATRIBUTO	PRODUCTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Instalación y Configuración	Instalador Windows	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3
	Distribución auto contenidad (excepto clientes SCV)	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Java como plataforma de ejecución	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Java como plataforma de proyecto	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Maven como herramienta de construcción	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	No requiere modificar los scripts de construcción	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Soporta múltiples proyectos	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Configuración automática desde script de construcción	✗ 0	✓ 3	✓ 3	0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
	Configuración Archivo Texto	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3
TOTAL		**21**	**24**	**15**	**24**	**15**	**24**	**24**	**24**	**21**	**18**	**27**	**24**

Figura 29: Valoración productos comerciales frente a Instalación y Configuración.

	PRODUCTOS											
ATRIBUTO	UrbanCode Build (AnthillPro)	Bamboo	easyCIS	Electric Flow	FinalBuilder Server	OpenMake Meister	OpenMake Mojo	Parabuild	Pulse	QuickBuild	TeamCity	Zed
Soporte Gestión de Código Fuente	52	21	15	32	34	48	48	39	12	22	39	15
Soporte Relacionado Gestión de Código Fuente	7	6	6	3	9	15	9	6	3	3	6	0
Gestión de la Construcción	39	36	39	48	21	54	39	36	34	27	45	26
Seguridad	18	15	9	12	9	12	12	9	9	6	12	7
Publicación	30	27	39	18	24	18	18	25	15	12	18	9
Interface Web	45	42	39	36	28	6	6	46	45	33	45	31
Herramientas de Construcción Directamente Soportadas	28	36	36	30	18	27	27	30	24	11	18	6
Integración con Gestores de Problemas e Incidentes	19	6	0	14	0	9	9	9	9	2	9	3
Integración con Herramientas de Prueba	30	18	6	18	3	12	12	18	9	4	21	0
Integración con IDEs	6	6	0	6	0	6	6	1	0	2	9	0
Integración Inspección de Código	4	6	0	0	0	0	0	6	6	0	6	9
API Administración Remota	9	6	9	6	3	9	9	9	3	1	9	3
Instalación y Configuración	21	24	15	24	15	24	24	24	21	18	27	24
TOTAL	**308**	**249**	**213**	**247**	**164**	**240**	**219**	**258**	**190**	**141**	**264**	**133**

Figura 30: Resumen valoración herramientas comerciales frente a sus atributos.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Soporte Gestión de Código Fuente	AccuRev	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0
	AlienBrain	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	BitKeeper	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓	✗ 0	✗ 0
	ClearCase	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0
	CA Harvest	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0
	CM Synergy	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0
	CVS	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0
	Dimensions	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	"File system SCM"	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0
	Git	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	ℹ 1	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0
	HTTP file	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0
	Mercurial	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0
	MKS	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0
	Perforce (p4)	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0
	PVCS	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓	✗ 0	✗ 0
	SourceGear Vault	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	StarTeam	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0
	Subversion	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Surround	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Team Foundation Server	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0
	VSS	⚠ 2	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0
	VSS Journal	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
TOTAL		**29**	**6**	**15**	**15**	**57**	**45**	**4**	**15**	**12**	**42**	**27**	**3**

Figura 31: Valoración productos libres frente al Soporte de Gestión de Código Fuente.

4.3 Tabulación de resultados de herramientas libres

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Gestión de la Construcción	Construcciones paralelas (abilidad para construir muchos proyectos de forma simultanea)	✓ 3	0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Construcciones Distribuidas	✓ 3	0	✓ 3	✗ 0	✓ 3	✗ 0	0	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3
	Ejecuta compiladores y enlazadores con Build Avoidance (construcciones incrementales)	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	0	✗ 0
	Acelera llamadas a compiladores y enlazadores a través de procesamiento multi-hilos	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	Agentes de auto actualización de código	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	0	0
	Realiza exploración de dependencias del código fuente, análisis de impacto y presentación de informes	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Auto generación del archivo build.xml basado en archivos de proyecto del IDE	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓	✗ 0	✗ 0	✗ 0	✗ 0
	Construcciones forzadas manualmente	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0

Figura 31: Valoración productos libres frente al Soporte Relacionado de Gestión de la Construcción (parte 1).

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Gestión de la Construcción	Gestión compilado de X-platform y X-language	0	0	3	0	0	0	0	3	3	0	0	0
	Construcciones disparadas por el SCV	3	0	3	3	3	0	0	0	3	3	3	3
	Construcciones basadas en encuentas al SCV	3	0	3	3	3	3	3	3	3	3	3	0
	Construcción con calendarios	3	0	3	0	3	3	0	3	0	3	3	0
	Promoción de Construcciones	1	0	3	0	0	0	0	3	3	3	3	0
	Dependencias entre proyectos	0	0	3	0	0	0	0	3	3	3	3	0
	Borrado de construcciones	1	0	3	3	0	0	0	3	0	3	3	0
	Reproduce historial de construcciones	1	0	3	3	0	0	0	0	3	0	3	0
	Proactivo (puede prevenir roturas de las construcciones)	0	0	0	0	0	0	0	3	0	0	0	3
	Detecta nuevas pruebas fallidas mientras se construye	0	0	3	0	0	0	0	0	0	3	0	0
	Notifica cuando la primera prueba falla en la construcción	0	0	3	0	0	0	0	0	0	0	0	0
TOTAL		**21**	**0**	**48**	**18**	**18**	**12**	**9**	**36**	**30**	**36**	**27**	**12**

Figura 33: Valoración productos libres frente al Soporte Relacionado de Gestión de la Construcción (parte 2).

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Soporte Relacionado Gestión de Código Fuente	Filtrado SCV	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0
	Múltiples SCV	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0
	Puede crear nuevos repositorios en el SCV	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0
	Huellas de los binarios construidos de regreso a SCV gestionados versiones de código	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✗ 0
	Audita archivos de origen utilizados en los binarios, pero no almacenados en un repositorio SCV	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	0	✗ 0	✗ 0	✗ 0	✗ 0
TOTAL		**0**	**0**	**6**	**6**	**6**	**6**	**0**	**6**	**12**	**3**	**3**	**0**

Figura 35: Valoración productos libres frente al Soporte Relacionado de Gestión de Código Fuente.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Seguridad	Autenticación de Usuario	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	Esquemas de autorización de usuario	✓ 3	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	Integración LDAP	ℹ 1	✗ 0	✗ 0	AD	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✗ 0
	Kerberos	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
	Single Sign On	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0
	JAAS Personalizado	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0	✗ 0
TOTAL		**7**	**0**	**0**	**6**	**3**	**0**	**0**	**9**	**9**	**9**	**9**	**0**

Figura 34: Valoración productos libres frente a la Publicación.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Publicación	Confluence	0	0	0	0	0	0	0	0	0	0	0	0
	Email	3	3	3	3	3	3	3	3	3	3	3	3
	Corre ejecutable		0	0	3	3	3	0	3	0	3	3	
	FTP	0	0	0	3	3	0	0	3	0	3	0	0
	IRC	3	0	3	0	0	0	0	3	0	3	0	0
	Jabber	3	0	0	0	3	0	0	3	0	3	3	0
	Lotus Sametime	0	0	0	0	3	0	0	0	0	3	3	0
	NetSend	0	0	0	3	0	3	0	3	0	0	0	0
	ProjectStart	0	0	0	0	0	3	0	0	0	0	0	0
	RSS	1	0	3	3	3	2	0	3		3	3	0
	SCP	0	0	0	0	3	0	0	3	0	3	0	0
	Windows System Tray	0	0	0	3	3	3	3	3	3	3	3	3
	Formatted Logging	0	0	0	3	3	3	0	3	3	0	3	0
	Yahoo Messenger	1	0	0	0	3	2	0	3	0	0	0	1
	MSN Messenger	3	0	0	0	0	0	0	3	0	0	3	0
	X10	0	0	0	3	3	3	0	3	0	0	0	0
TOTAL		**14**	**3**	**9**	**24**	**33**	**25**	**6**	**39**	**9**	**27**	**24**	**7**

Figura 36: Valoración productos libres frente a Instalación y Configuración.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Interface Web	Ver cambios	3	0	3	3	3	0	3	3	3	3	3	0
	Añadir nuevos proyectos	3	0	0	0	2	0	0	3	3	3		0
	Clonar proyectos	0	0	0	0	0	0	0	3	0	3	3	0
	Borrar proyectos	3	0	0	0	2	0	0	3	0	3	3	0
	Modificar proyectos	3	0	0	0	2	0	0	3	0	3	3	0
	Matar construcciones	1	0	3	0	3	0	0	3	3	3	3	0
	Pausar construcciones	1	0	0	0	3	3	0	0	3	0	3	0
	Acceso a construir artefactos	3	0	3	3	3	0	0	3	3	3	3	0
	Explorar la copia de trabajo de IC	3	0	0	0	0	0	3	0	0	3	0	0
	Elliminar una copia de trabajo de IC	3	0	0	0	0	2	0	0	0	3	0	0
	Búsqueda en construcciones	0	0	0	0	0	0	0	3	0	0	3	0
	Gráficos históricos	1	0	3	3	3	0	0	3	0	3	0	0
	Página web con actualización automática	3	0	3	3	3	3	3	3	3	3	3	3
	Soporte multi proyectos	3	3	0	3	3	3	0	3	3	3	3	0
	Vista multi project	3	0	0	3	3	3	0	3	3	3	3	3
	Añadir o remover agentes de equipo (para construcciones distribuidas)	0	0	0	0	0	0	0	3	3	3	0	0
TOTAL		**33**	**3**	**15**	**18**	**30**	**14**	**9**	**39**	**27**	**42**	**33**	**6**

Figura 37: Valoración productos libres frente a la Interface Web.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Herramientas de Construcción Directamente Soportadas	Script de línea de comandos o shell	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3
	Ant	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	Groovy	✗ 0	✗ 0	✓	✗ 0	✗ 0	✗ 0	✗ 0	✓	Via línea de comando	✓	✗ 0	✗ 0
	OpenMake Meister	✗ 0	✗ 0	✓ 3	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0
	Maven	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	Via línea de comando	✓ 3	✓ 3	✗ 0
	Maven2	✓ 3	✗	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✓ 3	Via línea de comando	✓ 3	✓ 3	✗ 0
	Make	✓ 3	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✗ 0	✓ 3	Via línea de comando	✗	✗ 0	✗ 0
	MsBuild	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	Via línea de comando	✓ 3	✗ 0	✗ 0
	NAnt	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0
	Rake (Ruby)	✗ 0	✗ 0	✓ 3	✗ 0	✗ 0	✗ 0	✓ 3	✓ 3	✓ 3	✓ 3	✗ 0	✗ 0
	Visual Studio ('devenv')	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	Via línea de comando	✗ 0	✗ 0	✗ 0
	FinalBuilder	✗ 0	✗ 0	✓ 3	✓ 3	✗ 0	✓ 3	✗ 0	✓ 3	Via línea de comando	✗ 0	✗ 0	✗ 0
TOTAL		**18**	**6**	**33**	**18**	**18**	**18**	**3**	**33**	**12**	**24**	**15**	**3**

Figura 38: Valoración productos libres frente a las Herramientas de Construcción Directamente Soportadas.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Integración con Herramientas de Prueba	Agitar	0	0	0	0	3	0	0	0		0	0	0
	CppUnit result rendering	0	0	0	0	0	0	0	0	3	3	0	0
	JUnit result rendering	1	0	0	3	3	3	0	0	3	3	3	0
	NUnit result rendering	0	0	0	3		3	0	3	3	3	0	0
	QualityCenter test rendering	0	0	0	0	0	0	0	0	3	0	0	0
	PHPUnit result rendering	0	0	0	0	0	0	0	0	3	3	0	0
	PMD result rendering	0	0	0	0	0	0	0	0	3	3	0	0
	Clover result rendering	0	0	0	0	0	0	0	0	3	3	0	0
	Selenium result rendering	0	0	0	0	0	0	0	0	3	3	0	0
	SilkCentral	0	0	0	0	0	0	0	0	3	3	0	0
	MSTest result rendering	0	0	0	0	0	0	0	3	0	0	0	0
TOTAL		**1**	**0**	**0**	**6**	**6**	**6**	**0**	**6**	**27**	**24**	**3**	**0**

Figura 39: Valoración productos libres frente a Integración con Herramientas de Prueba.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Integración con Gestores de Problemas e Incidentes	Bugzilla	0	0	0	0	0	0	0	0	3	3	0	0
	ClearQuest	0	0	0	0	0	0	0	0	0	0	0	0
	Confluence	0	0	0	0	0	0	0	0	3	0	0	0
	JIRA	0	0	0	0	0	0	0	0	3	3	0	0
	Mingle	0	0	0	0	0	0	0	0	3	0	0	0
	QualityCenter	0	0	0	0	0	0	0	0	0	0	0	0
	Rally	0	0	0	0	3	0	0	0	0	3	0	0
	Rubyforge.org	0	0	0	0	0	0	0	0	0	0	0	0
	Scarab	0	0	0	0	0	0	0	0	0	0	0	0
	Sourceforge.net	0	0	0	0	0	0	0	0	0	0	0	0
	Trac	0	0	1	0	0	0	0	0	3	3	0	0
	VersionOne	0	0	0	0	3	0	0	0	0	0	0	0
TOTAL		**0**	**0**	**1**	**0**	**6**	**0**	**0**	**0**	**15**	**12**	**0**	**0**

Figura 41: Valoración productos libres frente a Integración con Gestores de Problemas e Incidentes.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Integración Inspección de Código	Bonsai	1	0	0	0	0	0	0	0	0	0	0	0
	ViewVC	1	0	0	0	0	3	0	0	0	3	3	0
	Fisheye	1	0	0	0	0	0	0	0	0	3	3	0
TOTAL		**3**	**0**	**0**	**0**	**0**	**3**	**0**	**0**	**0**	**6**	**6**	**0**

Figura 40: Valoración productos libres frente a Integración Inspección de Código.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
API Administración Remota	REST	0	0	0	0	0	0	0	0	3	3	0	0
	SOAP	0	0	0	0	0	3	0	3	0	0	0	3
	XML-RPC	3	0	3	0	0	2	0	3	0	0	0	0
	JMX	0	0	0	0	3	0	0	0	0	3	0	0
	Hessian	0	0	0	0	0	0	0	0	0	0	3	0
	Biblioteca de cliente	0	0	0	.Net Remoting	0	0	0	3	0	0	0	0
TOTAL		**3**	**0**	**3**	**0**	**3**	**5**	**0**	**9**	**3**	**6**	**3**	**3**

Figura 42: Valoración productos libres frente a API Administración Remota.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Integración Inspección de Código	Bonsai	1	0	0	0	0	0	0	0	0	0	0	0
	ViewVC	1	0	0	0	0	3	0	0	0	3	3	0
	Fisheye	1	0	0	0	0	0	0	0	0	3	3	0
TOTAL		**3**	**0**	**0**	**0**	**0**	**3**	**0**	**0**	**0**	**6**	**6**	**0**

Figura 43: Valoración productos libres frente a Integración con IDEs.
Fuente: Elaboración propia.

ATRIBUTO	PRODUCTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Instalación y Configuración	Instalador Windows	1	0	3	3	3	3	3	3	3	3	3	3
	Distribución auto contenida (excepto clientes SCV)	3	0	3	3	3	3	3	0	3	3	3	3
	Java como plataforma de ejecución	3	0	0	0	3	0	0	0	3	3	3	0
	Java como plataforma de proyectos	3	0	3	0	3	0	0	0	3	3	0	0
	Maven como herramienta de construcción	3	0	3	0	3	0	0	0	0	3	3	0
	No requiere modificar los scripts de construcción	3	0	3	3	3	3	0	3	3	3	3	3
	Soporta múltiples proyectos	3	0	0	3	3	3	0	3	3	3	3	0
	Configuración automática desde script de construcción	3	0	0	0	0	0	0	3	0	3	0	0
	Configuración Archivo Texto	0	0	3	3	3	3	3	3	3	3	0	3
TOTAL		**22**	**0**	**18**	**15**	**24**	**15**	**9**	**15**	**21**	**27**	**18**	**12**

Figura 44: Valoración productos libres frente a Instalación y Configuración.
Fuente: Elaboración propia.

	PRODUCTOS											
ATRIBUTO	Apache Continuum	Apache Gump	BuildBot	CI Factory	Cruise Control	Cruise Control.NET	Cruise Control.rb	easyCIS	Go	Jenkins / Hudson	LuntBuild	Sin
Soporte Gestión de Código Fuente	29	6	15	15	57	45	4	15	12	42	27	3
Soporte Relacionado Gestión de Código Fuente	0	0	6	6	6	6	0	6	12	3	3	0
Gestión de la Construcción	21	0	48	18	18	12	9	36	30	36	27	12
Seguridad	7	0	0	6	3	0	0	9	9	9	9	0
Publicación	14	3	9	24	33	25	6	39	9	27	24	7
Interface Web	33	3	15	18	30	14	9	39	27	42	33	6
Herramientas de Construcción Directamente Soportadas	18	6	33	18	18	18	3	33	12	24	15	3
Integración con Gestores de Problemas e Incidentes	0	0	1	0	6	0	0	0	15	12	0	0
Integración con Herramientas de Prueba	1	0	0	6	6	6	0	6	27	24	3	0
Integracióncon IDEs	2	0	0	0	6	0	0	0	0	6	3	0
Integración Inspección de Código	3	0	0	0	0	3	0	0	0	6	6	0
API Administración Remota	3	0	3	0	3	5	0	9	3	6	3	3
Instalación y Configuración	22	0	18	15	24	15	9	15	21	27	18	12
TOTAL	**153**	**18**	**148**	**126**	**210**	**149**	**40**	**207**	**177**	**264**	**171**	**46**

Figura 45: Resumen valoración herramientas libres frente a sus atributos.
Fuente: Elaboración propia.

4.4 Análisis de resultados

De acuerdo con los resultados obtenidos en la valoración de los atributos de los productos comerciales y libres se pudo distinguir que trascienden las herramientas de tipo comercial. El promedio alcanzado por la herramientas comerciales es de 218.83 puntos mientras que las herramientas de uso libre alcanzan un promedio de 142.41 puntos.

Entre los dos grupos de herramientas, despuntan claramente un subgrupo reducido y en cada uno de ellos hay una que se muestra como la de mejores características. Estas herramientas son, para los productos comerciales UrbanCode Build y, para los productos de uso libre Jenkins / Hudson. Se debe hacer notar también que Jenkins / Hudson es la segunda en puntaje con respecto a todas las herramientas involucradas en la evaluación.

Trasladando los resultados obtenidos en la valoración de atributos a la tabla de relación entre los atributos y los parámetros podemos ver los resultados en función de esas características de calidad.

La figura 46 muestra el resultado de la valoración de la herramienta comercial que mayor puntaje logró mientras que, la figura 47 muestra el resultado de la valoración de la herramienta de uso libre.

A continuación en la tabla 4-33 podemos notar que en todos los parámetros considerados UrbanCode Build superar a Jenkins, de igual forma, en la figura 46 se aprecia que con los valores ponderados se mantiene esa diferencia.

ATRIBUTOS	PARÁMETROS			
	Operabilidad	Adecuación	Exactitud	Madurez
Soporte Gestión de Código Fuente		52		
Soporte Relacionado Gestión de Código Fuente				7
Gestión de la Contrucción	39			
Seguridad				18
Publicación	30			
Interface Web	45			
Herramientas de Construcción Directamente Soportadas	28			
Integración con Gestores de Problemas e Incidentes		19		
Inegración con Herramientas de Prueba				30
Integración con IDEs		6		
Integración Inspección de Código			4	
API Administración Remota			9	
Instalación y Configuración	21			
	163	**77**	**13**	**55**

Figura 47: Valoración de parámetros de calidad de software para UrbanCode Build.

ATRIBUTOS	PARÁMETROS			
	Operabilidad	Adecuación	Exactitud	Madurez
Soporte Gestión de Código Fuente		42		
Soporte Relacionado Gestión de Código Fuente				3
Gestión de la Contrucción	36			
Seguridad				9
Publicación	27			
Interface Web	42			
Herramientas de Construcción Directamente Soportadas	24			
Integración con Gestores de Problemas e Incidentes		12		
Inegración con Herramientas de Prueba				24
Integración con IDEs		6		
Integración Inspección de Código			6	
API Administración Remota			6	
Instalación y Configuración	27			
	156	**60**	**12**	**36**

Figura 46: Valoración de parámetros de calidad de software para Jenkins.

Tabla 5:
Resultados por parámetros de evaluación.

	PARAMETROS DE EVALUACIÓN				
PRODUCTO	Operabilidad	Adecuación	Exactitud	Madurez	**TOTAL**
UrbanCode Build	163	77	13	55	**308**
Jenkins / Hudson	156	60	12	36	**264**

Tabla 6:
Resultados ponderados.

	PARAMETROS DE EVALUACIÓN				
PRODUCTO	Operabilidad	Adecuación	Exactitud	Madurez	**TOTAL**
UrbanCode Build	30,5625	33,6875	0,8125	17,1875	82,25
Jenkins/Hudson	29,25	26,25	0,75	11,25	67,5

Para el caso de los productos comerciales las herramientas que despuntan son aquellas respaldadas por grandes corporaciones como es el caso de UnbanCode Build anteriormente conocida como Anthill y que fue adquirida por IBM. Y para el caso de los productos de uso libre tenemos herramientas que están respaldadas por grandes comunidades como por ejemplo Apache Software Foundation en donde tenemos Gump y Continuum.

4.5 Informe de Análisis

El análisis realizado en este trabajo parte con algunas ideas sobre la gestión del ciclo de vida de las aplicaciones y una rápida comparación entre las metodologías de desarrollo tradicionales y las conocidas como metodologías ágiles, estas últimas, se basan en un proceso iterativo a diferencia de las tradicionales que, como la metodología en cascada tiene fases bien delimitadas que si bien le presentan como robusta, también le dan una característica de rigidez o poco flexible.

También se describe la situación por la que ha atravesado el Banco Central del Ecuador y específicamente la Coordinación General de Tecnologías de la Información y Comunicaciones en lo que respecta al desarrollo de software y, en particular los problemas encontrados en el desarrollo de productos dirigidos a presentarse al usuario a través de tecnologías Web.

Se ha identificado un conjunto de tareas repetitivas en la fase de desarrollo de los proyectos que son candidatas a ser automatizadas. También se introdujo la idea de aprovechar la automatización de las tareas para incorporar herramientas que ayuden a mejorar la calidad del software en desarrollo y visibilizar la evolución del mismo a través de herramientas de seguimiento de incidentes.

De acuerdo con los objetivos, se ha logrado describir los tipos de herramientas que permitirían configurar un entorno de Integración Continua o ecosistema de desarrollo de software orientado a implementar el modelo informático de Integración Continua y, se han considerado un conjunto de las mismas tanto en el ámbito comercial como en el libre para ser evaluadas en base a un conjunto de parámetros y sub-parámetros de calidad del software y un conjuntos de atributos específicos de acuerdo al tipo de herramienta en este caso, un motor o servidor de Integración Continua.

Como resultado de la evaluación se ha encontrado que las herramientas que mejor se ajustan a los criterios considerados son, UrbanCode Build dentro del conjunto de herramientas comerciales y Jenkins / Hudson en el conjunto de herramientas de uso gratuito.

CAPÍTULO 5

5 CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

En base a lo expuesto en los capítulos anteriores de este trabajo, se han obtenido las siguientes conclusiones:

- Existen metodologías como el Desarrollo Dirigido por Pruebas y prácticas recomendadas por las Métodos Iterativos como las metodologías Ágiles y el Proceso Unificado dentro de los cuales destacan las prácticas de la Integración Continua.
- Existen herramientas de uso gratuito que permiten implementar la Integración Continua pero, las que destacan son las de tipo comercial puesto que contemplan prácticas que van más allá de la Integración Continua.
- Las herramientas que destacan son, dentro del grupo de las herramientas de uso gratuito, Jenkins y dentro del grupo de las herramientas comerciales UrbanCode (IBM) aunque en este mismo grupo se han identificado otras que también cubren las expectativas establecidas.
- Cualquier organización en donde se haga desarrollo de software debería contemplar el adoptar del modelo de Integración Continua, especialmente aquellas en las que se trabaja con lenguajes de programación compilados como son los de las plataformas Java y .Net.
- El modelo informático de Integración Continua es una forma de desarrollo que mejora el proceso de pruebas y puesta en producción de un producto de software nuevo o en mantenimiento. La implementación de una herramienta

que permita la automatización de varias tareas que eventualmente son pesadas y repetitivas permitirá que los desarrolladores se centren en asegurarse que el código haga lo que tiene que hacer.

- Dentro de lo que es el proceso de desarrollo de software, las fases que deberán enfrentar el mayor reto son las de construcción y pruebas, en esa línea, las áreas de la organización que deberán asumir ese reto son Operaciones y Control de Calidad.
- Es previsible que el impacto del Control de Calidad sea mayor mientras más temprano sea realizado, es por eso que, se esperaría que las pruebas y el control de calidad realizado por medio de herramientas que se involucran en el modelo de Integración Continua permitan lograr productos con menos defectos y de mejor calidad.
- Las herramientas de Integración Continua más populares ofrecen un conjunto de características similares y, la selección de la herramienta correcta dependerá de muy específicas características, condiciones o requerimientos. Pero cualquiera que se escoja debería ejecutar las tareas más importantes.

5.2 Recomendaciones

Además de considerar los parámetros y sub-parámetros de calidad identificados en el desarrollo de este trabajo, al momento de decidir si escoger una herramienta de tipo comercial o de uso gratuito conviene también tomar en cuenta factores como él grado de madurez de la organización. Oscar Berta-Hinostroza en (BERTA-HINOSTROZA, 2011), propone un “Cuestionario para medir la Integración Continua” con el cual se podría tener una aproximación a la situación desde la que se arrancaría un proceso de adopción del modelo de Integración Continua.

Si la organización está iniciando en la adopción de metodologías ágiles y los procesos no están aún maduros, quizá no haga falta más que herramientas de uso gratuito como Jenkins pero, si ha alcanzado un nivel de madurez en el que los procesos están claramente establecidos y las buenas prácticas son cosa de todos los días, quizá sea tiempo en considerar en invertir en una herramienta comercial como UrbanCode Build.

Para el caso del BCE y la Dirección de Desarrollo Informático se recomienda implementar la herramienta Jenkins y a través de su uso fomentar la incorporación de prácticas que permitan gradualmente introducir el modelo de Integración Continua.

BIBLIOGRAFIA

BECK, K. (2001). *Manifiesto Ágil*. Recuperado el marzo de 2015, de http://agilemanifesto.org/iso/es/

BERTA-HINOSTROZA, O. (1 de noviembre de 2011). INCORPORACIÓN DE LA INTEGRACIÓN CONTINUA EN EL DESARROLLO DE SOFTWARE: CASO DE ESTUDIO: ORGANISMO SUPERVISOR DE LA INVERSIÓN EN ENERGÍA Y MINERÍA. Piura.

BOEHM, B. W. (2002). *Software engineering economics.* New York.

CAÑADILLAS MEDINA, D. (2010). *IMPLANTACIÓN DE ARQUITECTURA DE DESARROLLO ÁGIL*. Madrid: Escuela Politécnica Superior.

DIETRICH, J. (22 de septiembre de 2011). *DZone.* Recuperado el 1 de febrero de 2015, de https://dzone.com: https://dzone.com/articles/dependency-analysis-and-1

EL DIRECTORIO DEL BANCO CENTRAL DEL ECUADOR. (14 de enero de 2014). *BANCO CENTRAL DEL ECUADOR.* Recuperado el 01 de febrero de 2015, de http://www.bce.fin.ec/documents/pdf/general/estatutoOrganico.pdf

EXPÓSITO, R. (28 de diciembre de 2009). *Raúl Expósito.* Recuperado el febrero de 2015, de http://raulexposito.com/: http://raulexposito.com/documentos/analisis-estatico-codigo/

FOWLER, M. (01 de mayo de 2006). *Martin Fowler*. Obtenido de Continuos Integration: http://www.martinfowler.com/articles/continuousIntegration.html

FUNDIBEQ. (2 de noviembre de 2007). *FUNDIBEQ.* Obtenido de http://www.fundibeq.org: http://www.fundibeq.org/opencms/export/sites/default/PWF/downloads/gallery/methodology/tools/diagrama_de_pareto.pdf

GARCÍA DÍAZ, V. (2011). *MDCI: Model-Driven Continuos Integration.* UNIVERSIDAD DE OVIEDO, Departamento de Informática. Oviedo: UNIVERSIDAD DE OVIEDO.

HOLCK, J., & JORGENSEN, N. (2003). *Australasian Journal of Information Systems.* Obtenido de http://journal.acs.org.au/index.php/ajis/article/view/145/125

JACOBSON, I. (1 de julio de 2007). *The Journal of Object Technology*. Obtenido de http://www.jot.fm/issues/issue_2007_07/column5.pdf

McCONNELL, S. (2004). *Code Complete, Second Edition* (2 ed.). Redmond, WA, USA: Microsoft.

OGC, O. o. (1996). *https://www.prince2.com*. Recuperado el Febrero de 2015, de Prince2.com: https://www.prince2.com/uk/what-is-prince2

ONGEI. (27 de mayo de 2004). Guía Técnica sobre Evaluación de Software en la Administración Pública. Lima, Perú.

ORIENTE, J. (17 de agosto de 2003). *Formandobits.* Obtenido de 10 preguntas que te debes hacer antes de implantar una nueva herramienta en tu organización: http://formandobits.com/2013/08/10-preguntas-a-hacerse-antes-de-implantar-una-nueva-herramienta-en-tu-organizacion/

PAE. (2001). *http://administracionelectronica.gob.es*. Obtenido de Portal de Administración Electrónica: http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

PRESSMAN, R. S. (2010). *Ingeniería del software. Un enfoque práctico* (Séptima edición ed.). México, D.F., México, D.F., México: McGrawHill.

RECENA, M. J. (9 de agosto de 2008). *MI ESPACIO*. Obtenido de ¿QUÉ ES UN ECOSISTEMA SOFTWARE?: http://manuelrecena.com/blog/archives/219

SCHWABER, Ken; SUTHERLAND, Jeff;. (Julio de 2013). La Guía Definitiva de Scrum: Las Reglas del Juego.

SHARPE, R. (01 de abril de 2003). *The Economist.* Obtenido de Building a better bug-trap: http://www.economist.com/node/1841081

SOGETI Innovation Today. (04 de marzo de 2014). Obtenido de http://itblogsogeti.com/2014/03/04/la-gestion-de-la-deuda-tecnica-carlos-mendible-sogeti/

TASSEY, G. (01 de mayo de 2002). *National Institute of Standard and Technology.* Obtenido de http://www.nist.gov/: http://www.nist.gov/director/planning/upload/report02-3.pdf