Congreso Nacional de Software Libre 2004

¿Por Qué Fracasan los Proyectos de Software?; Un Enfoque Organizacional

J. Jesús María Zavala Ruiz
Postgrado en Estudios Organizacionales
Universidad Autónoma Metropolitana-Iztapalapa
Mexico, D.F.
Tel. +52 (55) 5841-9218
e-mail: jzavalar@yahoo.com

Abstract

Los proyectos de software son proyectos que tienen características muy particulares: sus entregables son digitales (programas de cómputo, archivos fuente, diagramas, modelos, manuales, etc. digitales), requieren mucha creatividad en la mayoría de sus fases, con tasas de fracaso de más del 70% (según el Standish Group). Los proyectos representan el modelado parcial de la organización, son de gran complejidad, son muy costosos, son manejados con poca experiencia administrativa y son de gran importancia para las organizaciones en la sociedad postindustrial.

En este ensayo se explora el panorama de los proyectos de software, su importancia en la sociedad y en la organización. Posteriormente, se hace un análisis de los factores que determinan el éxito o fracaso de los proyectos de software. Finalmente se analizan las causas de la falla de proyectos desde la perspectiva de los estudios organizacionales y se propone un enfoque para reducir esa tasa de fracaso mediante la administración de proyectos, la ingeniería de software y el aprendizaje organizacional.

Keywords: software projects, project failure, project success, software crisis, organizational approach, software engineering, project management, organizational learning, systems analysis, unified modeling language, UML, organizational studies, organization theory, organizational studies

Keywords: proyectos de software, falla de proyectos, exito en proyectos, crisis del software, enfoque organizacional, ingeniería de software, administración de proyectos, aprendizaje organizacional, analisis de sistemas, lenguaje de modelado unificado, UML, estudios organizacionales, teoria de la organización, estudios organziacionales

Introducción

Este ensayo aborda el problema más importante de la industria de software y por consecuencia, de las organizaciones: el fracaso de los proyectos de software. Se abordan los conceptos mínimos para aquellos lectores que no son conocedores del tema y se desarrolla una propuesta teórica sobre el fracaso de los proyectos de software. La importancia del software es tal que hoy prácticamente todas las organizaciones dependen en mayor o menor medida del software para operar. El software es cada vez más importante en la vida de pública y privada de todos. El software está en las computadoras, en el horno de microondas, en los juguetes, en todos los aparatos y equipos modernos. El comercio electrónico será pleno en un futuro muy cercano. El desarrollo de software de hoy se puede

caracterizar más como un actividad artesanal que como una disciplina de ingeniería.

Conceptos básicos

Antes que nada se van a aclarar algunas definiciones clave para este ensayo.

Datos "son los hechos brutos acerca de la organización y sus transacciones de negocios. La mayoría de los datos tienen poco significado uso por sí mismos". *Información* "son los datos que han sido refinados y organizados mediante el procesamiento y con un sentido determinado". *Conocimiento* "es la utilización completa de información y datos, junto con el potencial de las habilidades, competencias, ideas, intuiciones, compromisos y motivaciones de la gente" (Según Grey, citado por [Zavala 2003]. *Conocimiento corporativo* "es el cuerpo colectivo de experiencias y entendimiento de los procesos de una organización para administrar situaciones planeadas y no planeadas" [*op.cit.*].

Sistema es "un grupo integrado de elementos para cumplir un objetivo definido. Estos incluyen hardware, software, firmware, gente, información, técnicas, facilidades, servicios y otros elementos de soporte" ([INCOSE 1998], citado por [Zavala 2003]). Más específicamente, un *sistema de información* (SI) es "el conjunto de personal, datos, procesos, interfases, redes y tecnología que interactúan con propósito de soportar y mejorar las operaciones diarias de un negocio" [ídem]. Un *sistema de cómputo* o *aplicación de cómputo* "es una solución automatizada para uno o más problemas y necesidades de negocios" [*op.cit.*]. Como puede apreciarse, en un sistema de información hay uno o más sistemas de cómputo.

Tecnología de Información (*Information Technology*) o TI "es un término contemporáneo que describe la combinación de la tecnología de cómputo (hardware y software) con la tecnología de telecomunicaciones (redes de datos, imágenes y voz)" [*ídem*]. *Hardware* son los componentes mecánicos, magnéticos, electrónicos, y eléctricos que integran una computadora[1]. Según la [IEEE 1990] el *software* es "la suma total de los programas de cómputo, procedimientos, reglas y documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo". El software "es aquello invisible y etéreo que hay en cada sistema de información" [Zavala 2003b]. Un *producto de software* es "un producto diseñado para entregarse a un usuario, comúnmente se le conoce como paquete, aplicación o sistema de cómputo" [*ídem*].

Ciclo de vida del software es modelo que define el proceso de desarrollo del software, desde su concepción hasta su puesta en operación y mantenimiento. *Proceso de desarrollo* es el modelo mediante el cual de define *qué* se debe realizar, *quién* las debe hacer, *cuando* deben comenzar y terminar y finalmente *cómo* alcanzar un determinado objetivo. Estos dos modelos son fundamentales ya que de ellos dependerán los enfoques que se abordarán para resolver la complejidad de este proceso [Zavala 2003].

Como podemos estimar, el desarrollo de software que requieren las organizaciones solo es factible desarrollarlo mediante una organización. *Desarrollador* es el profesional de las ciencias de la computación[2] que se aboca a desarrollar software. *Ingeniero de software* es el profesional del desarrollador de software que abarca distintos roles como analista, arquitecto,

[1] http://www.thefreedictionary.com/computer%20hardware

[2] Al respecto Peter Denning hace un análisis muy exhaustivo sobre las características de la, ahora llamada, Profesión en Tecnologías de Información y el reto del profesional ante la sociedad [Denning 2001].

programador, probador, documentador, administrador de producto y líder de proyecto, entre otros.

Importancia del software

Hoy en día, el software forma parte de nuestra vida cotidiana e influye sobre muchas de las actividades que realizamos. Prácticamente hoy en día nadie, en el mundo permanece inmune al software. El software no se encuentra únicamente en las computadoras de nuestras casas y oficinas. De hecho, la calefacción, el horno de la cocina, el despertador digital, el equipo de audio, la televisión, los juguetes, los aviones, los ferrocarriles y prácticamente todo lo que se pueda imaginar, tan elemental como la lavadora, ya incorporan software que asegura su funcionamiento. La considerable importancia del software está en innumerables productos de todo tipo y además, ya puede considerarse un motor económico de creciente importancia. No solo en las redes de telecomunicaciones modernas que utilizan sistemas de telecomunicación el software juega un papel importante, sino que se estos sistemas controlados por software son capaces de hacer realidad la disponibilidad de servicios de alta calidad en plazos de tiempo y con un costo razonables.

El software no se elabora como cualquier producto industrial, no se construye, se desarrolla e involucra intensamente a la gente al ser una actividad mental y creativa [Zavala 2003b]. El software es complejo sencillamente porque está modelando la operación de la organización, y por lo mismo, se vuelve más complejo en tanto más compleja es la organización. La complejidad es precisamente la propiedad más importante del software. Por otro lado, el software nace a partir de la organización, evoluciona con ella y muere con ella. El software muere cuando ya no se le da mantenimiento. ¿Pero qué función tiene el software en la organización? Utilizando una analogía, se puede decir que el software es "lo invisible e inmaterial, la fuerza o la energía" que hace que las computadoras automaticen la operación de la organización. He aquí el poder del software: la automatización.

En menos de 25 años, entre la aparición de las primeras computadoras personales y hoy, ha tenido lugar la explosión de la industria del hardware. Según [Del Toro 2003] "en 1981, por ejemplo, la IBM PC 5150, una de las pioneras del cómputo personal, contaba con apenas 64 Kb[3] de memoria de acceso aleatorio (RAM), procesador de 4.77 Mhz[4], almacenaba su información en casettes y no ofrecía disco duro ni unidad de diskette." Hoy, una PC estándar ofrece varios miles de veces más que esos recursos, todo conforme a la implacable Ley de Moore[5]. Este fenómeno provoca la obsolescencia de las tecnologías de información en un tiempo relativamente breve (dos a 5 años) en comparación con otras industrias. Y por lo visto Intel intentará que la *Ley de Moore* continúe prevaleciendo algunas décadas por venir, ahora con la fotónica de silicio y la explosión de Internet [Panicc 2003] donde se unen las tecnologías de cómputo y comunicaciones que proveen una capacidad de operación nunca antes vista.

Al contrario del hardware, "numerosos expertos coinciden en que los próximos lustros verán,

[3] Kb, abreviación de Kilobytes, que equivalen a 1024 bytes, octetos o caracteres.

[4] Mhz, abreviatura de Megahertz (10^6 Hertz), unidad de medida de la frecuencia en ciclos por segundo.

[5] Gordon Moore pronosticó el rápido ritmo de las innovaciones tecnológicas en 1965. Hoy, la *Ley de Moore* sigue siendo válida. La Ley de Moore establece que el número de transistores disponibles para construir o poblar un circuito integrado de silicio se duplica cada dos años. Lograr este crecimiento exponencial en la densidad de los transistores requiere que el tamaño de los transistores se reduzca cada vez más. A su vez, esta constante reducción trae como resultado menores costos y un mayor rendimiento de los dispositivos de silicio construidos con estos transistores de menor tamaño [Intel 2003]

paralelamente, el gran auge del software, un fenómeno que comenzó a atisbarse en los años 80" [Del Toro 2003]. "La IBM PC 5150 ejecutaba un rudimentario sistema operativo, desarrollado justamente por Gates, que sería el precursor del MS-DOS. Las PC de hoy en día ejecutan, en su gran mayoría, alguna versión de Windows" [*ídem*]. Este escenario en los años 80's también provocó varios fenómenos interesantes e insólitos en la industria del software: el monopolio de una sola empresa (Microsoft), el establecimiento de la licencia de uso de un producto digital (el software, "inventada" años antes por IBM) y la garantía del software (concebida única y exclusivamente como la reposición del medio) que, curiosamente *no* garantiza la calidad del producto (el software) y que todos los consumidores han aceptado de tajo sin protestar.

Otro suceso que ha provocado una revolución organizacional es la explosión comercial de Internet, la red de redes mundial a mediados de los 90's. En 1989 había 100,000 hosts[6] para 1990 había tres veces más. En 1990 la ARPANET del Departamento de Defensa (DoD) de los Estados Unidos deja de operar y entra en funcionamiento la NSFNet en conjunto de varias redes comerciales. También se apoya con $3,000 millones de dólares la investigación en los Estados Unidos para desarrollar la Internet en una herramienta comercial viable y en esta iniciativa el Senador Al Gore acuña el término de la *supercarretera de la información*. En 1993 había un millón de hosts en Internet y para 2002 llegó a 140 millones. Los usuarios suman más de 250 millones [Zakon 2003].

Por último, un fenómeno que socialmente y económicamente no ha sido aprovechado en su potencial por las organizaciones es el llamado software libre (*free software*) y el software de código abierto (*open source software*) que desde su surgimiento y su actual crecimiento explosivo ha impulsado mucha de la tecnología que hoy soporta el propio crecimiento y operación de la misma Internet. Académicamente tampoco se le ha dado la importancia que tiene, tanto en aspectos docentes, como en la propia generación de conocimiento.

Hoy en día, lo que *"se mueve"* son datos que se interpretan para obtener información y generar conocimiento organizacional, es decir, la capitalización de la experiencia del personal y los procesos de trabajo al interior de la organización. Es este principio de siglo está en proceso una revolución en la operación de las organizaciones afectadas por el desarrollo tecnológico (principalmente de las TI), su integración en cadenas productivas (y la integración de sus sistemas de información), el cambio de los paradigmas de la coordinación de actividades (facilitado por las comunicaciones), el teletrabajo (desde reuniones virtuales hasta supervisión y producción intelectual de productos digitales fuera de la oficina tradicional) y otros fenómenos organizacionales. Todo esto es posible gracias al software.

El Software en las Organizaciones

Desde siempre, el software se ha utilizado para operar las organizaciones, sea cual sea su ramo. En un principio, únicamente las organizaciones militares, hoy es de uso generalizado y ha invadido la esfera de las actividades personales. En este sentido podemos concebir al software como *la implementación moderna de las reglas, políticas y procedimientos de negocios*, en un sentido amplio. Toda operación importante queda registrada por el software en el sistema de información de la organización.

[6] Host = Servidor o equipo conectado a la red.

El software es el activo más importante de las organizaciones, aunque pocas veces se le otorgue valor distinto al de uso. El software se ha convertido en el corazón de la operación de la organización y prácticamente no hay algo que no esté relacionado con su operación, aunque en muchas organizaciones las actividades informáticas son consideradas una actividad "de apoyo". El software es tan importante que una *falla del software* puede paralizar a la organización entera y a sus socios de negocios. Todo crecimiento de la organización implica un crecimiento del software y de los requerimientos para su desarrollo. La factibilidad de implementación de las políticas de negocio pasa a depender de la capacidad del software y del personal que lo opera de gestionar los *datos* acorde a ella. Por ejemplo, ¿qué ocurre cuando se acude a una sucursal bancaria y "no hay sistema" - situación además, muy usual?... Pues sencillamente, es imposible hacer alguna operación, con las consecuencias pérdidas económicas. ¿Cuántos bancos hoy son más ineficientes en sus sistemas al atender al público, después de las fusiones de la última década?...

El *sistema de información* o comúnmente *sistema* de una organización es la representación parcial (o central) de la estructura y operación de la propia organización. Es una representación parcial porque hay aspectos que no se registran y probablemente nunca se registren en los sistemas de información como la toma de decisiones. Por ello es primordial asegurar que funcione correctamente. A medida que la organización crece, también los sistemas crecen en tamaño, funcionalidad, complejidad y rigidez. Los *parches* en sentido real terminan convirtiendo al sistema en un *Frankestein*. El software de una organización se convierte *de facto* en su activo más valioso y que sin embargo, normalmente no se considera como tal. Por ejemplo, ¿qué pasa si en una organización cualquiera se destruyen los datos sobre sus clientes, sus operaciones o su contabilidad? Eche a volar su imaginación un poco…

El impacto que tienen las TI es muy alto. La información, la comunicación, la coordinación y el software son los componentes importantes de la llamada "nueva economía", que algunos han dado en llamar la *webonomía*. Algunas organizaciones consideran a sus proyectos de TI como estratégicos y están alterando su estructura y operación, intentando adaptarse a la turbulencia del mercado y aprovechando las ventajas en la coordinación, el registro y análisis de información y el apoyo para la toma de decisiones que brindan los sistemas de información, logrando una ventaja competitiva. Otras organizaciones, sencillamente están utilizando las TI de manera colateral, aprovechando parcialmente las posibilidades de automatización en el flujo de información y procesos productivos; esencialmente operando como antes y no aprovechan la oportunidad para re-definir sus procesos de producción.

La Industria del Software

[Graham 2000] resume la historia de la industria del software en cuatro eras bien diferenciadas:

La *primera era* (1945-65) cubre el desarrollo y la comercialización inicial de la computadora. El software como actualmente se conoce no existe. Aún después que desarrollados los conceptos de *programa almacenado*, el software fue fundamentalmente desarrollado a la medida para computadoras muy grande y costosas (*mainframes*), básicamente en organizaciones militares y académicas. Durante los años 1950s emerge la adopción de arquitecturas de cómputo estándar soportadas por la emergencia de software que puede operar en un tipo de computadora.

Los modelos IBM 650 y posteriormente IBM 360 dominaron el mercado. Durante este periodo la mayor parte del software para computadoras es producido por los mismos fabricantes y por los propios usuarios.

Durante la *segunda era* (1965-1978) comienzan a aparecer los vendedores del software independientes (*independent software vendors* ISV). A finales de los años 1960s los fabricantes de computadoras mainframes separan sus *productos de software* de sus productos de hardware, separando el precio y la distribución de ambos. Comienzan a aparecer despachos de servicios de cómputo que proveen servicios de operación y soluciones de programación. Los usuarios de computadoras mainframes desarrollan *expertise* en soluciones para sus necesidades de aplicaciones y sistemas operativos.

Durante la *tercera era* (1978-93) el desarrollo y difusión de la computadora de escritorio o personal (PC) produce un crecimiento explosivo en la industria del software. Nuevamente, los Estados Unidos son el principal promotor de esa transformación y surge rápidamente el mercado del *software empaquetado*. La rápida adopción de la computadora de escritorio en los Estados Unidos es soportado por la emergencia de unos pocos diseños dominantes y la creación del *primer mercado masivo de software empaquetado*, los famosos paquetes o paquetería que enseñan en las escuelas comerciales en México. Los ISVs que entraron a la industria del software de escritorio se convierten en productores líderes de software y los ISVs de mainframes y minicomputadoras se ven reducidos. La rápida difusión del hardware de escritorio de bajo costo en combinación con la emergencia de unos pocos modelos de diseños dominantes para esta arquitectura merma la integración vertical entre los productores del hardware y software dando grandes oportunidades para los ISVs. La reducción en costo de la tecnología de cómputo expandió continuamente el potencial de aplicación de las computadoras.

La *cuarta era* del desarrollo de la industria del software (1994-presente) ha sido dominada por el crecimiento de la red y la masificación de las computadoras de escritorio dentro de las empresas en redes locales (LANs), de área amplia (WANs) e Internet, con servidores y *granjas* de servidores y millones de usuarios de Internet. Las redes han abierto oportunidades a la emergencia de nuevos segmentos de mercado del software. Algunas aplicaciones de red tiene el rápido crecimiento, tales como el *world wide web*[7] que utiliza html de manera este iba en todas las plataformas y la emergencia y empuje de las redes inalámbricas (*wireless*).

Las empresas estadounidenses han mantenido las posiciones dominantes en estos mercados. Hoy en día ya nadie puede dudar que el crecimiento explosivo de Internet está alterando estructuralmente la operación personal, organizacional y en general, la economía a tasas nunca antes vistas: creando canales de bajo costo para la distribución y comercialización de software empaquetado reduciendo las barreras de entrada a la industria; la posibilidad de la rápida penetración de los mercados por un producto de software, terrible un *killer app*[8].

Surge también un tipo de software diferente: el software libre (*free software*) y el software de código abierto (*open source software*), conocido en conjunto como FLOSS con tecnologías ampliamente usadas

[7] Plataforma de cómputo gráfica universal que facilita el uso del Internet con sus múltiples aplicaciones más allá de las más comunes como servidores web, correo-electrónico, telefonía sobre IP, entre las más conocidas.

[8] Una aplicación que se convierte en una aplicación que arrasa a sus competidoras por su funcionalidad fuera de serie.

como el sistema operativo Linux, el sistema operativo más seguro el OpenBSD, el servidor web Apache, el servidor de nombres de dominio (DNS) Bind que soportan el crecimiento y operación de Internet. El FLOSS está representando un nuevo paradigma del desarrollo de software y un nuevo paradigma para el equipamiento tecnológico y el desarrollo de una ventaja competitiva en las organizaciones que puede romper el monopolio de la multinacional Microsoft y que de hecho ésta considera como su principal amenaza comercial[9].

Como puede verse en este pequeño esbozo de la industria del software, ésta es una industria emergente en proceso de consolidación. Sus procesos de producción se encuentran inmersos en un ambiente de rápido cambio (igual que las organizaciones) y mucha presión. Desgraciadamente, la calidad del software no tiene una trayectoria tan brillante como su crecimiento, como más delante se verá, incluso la garantía del fabricante es extremadamente limitada en comparación con otro tipo de bienes o productos industriales.

Los Proyectos de Software

La mayor parte de los proyectos de software se desarrollan por equipos de desarrollo del proyecto de unas cuantas personas hasta grandes grupos de varias decenas de individuos. Son personal altamente especializado y calificado con actividad prácticamente intelectual y creativa. Prácticamente todos tienen la intención de automatizar parcialmente a la organización.

La percepción común que se tiene de la industria del software es que es una industria sólida pero sorprendentemente, no se caracteriza por la alta calidad generalizada de sus productos y servicios. La investigación que más se cita sobre el estado de los proyectos de software es el famoso "Reporte Caos" del Standish Group [Standish 1995] que prácticamente todos los investigadores asumen como la referencia obligada. El Standish Group [Standish 1998] clasifica los proyectos en tres tipos:

- Exitoso (*Successful*) – El proyecto se completa en tiempo y dentro del presupuesto, con todas las características y funciones
- Desafiante (*Challenged*) – El proyecto se completa y es operacional, pero más allá del presupuesto, más allá del tiempo estimado y con pocas de las características y funciones que fueron especificadas inicialmente
- Fracasado (*Failed*) – El proyecto es cancelado antes de completarse

Es indudable que los proyectos de software se caracterizan por altas tasas de fracaso o falla. Y a pesar de eso, el mito de que la industria es *alta tecnología* persiste. Y digo que es un mito porque la percepción de la mayoría de las organizaciones, incluso algunas de este mismo sector y del usuario común es que consideran que desarrollar software es una tarea fácil, entre otras cosas por ser un *producto digital*.

El Standish Group [ídem] en un seguimiento del *Chaos Report* encontró que en 1998 "en los Estados Unidos se gastaron más de $250,000 millones de dólares por año en el desarrollo de aplicaciones de TI en aproximadamente 175,000 proyectos. El costo promedio del desarrollo de un proyecto para una

[9] Para una referencia más completa revisar el *Manifiesto del Software Libre* de la comunidad mexicana de software libre (http://manifiesto.cofradia.org), el proyecto *GNU* y el esquema de licenciamiento *GPL* de la *Free Software Foundation* (http://www.fsf.org) y el proyecto *Open Standard Initaitive (OSI)* (http://www.opensource.org) y la excelente argumentación a favor del FLOSS de Wheeler, D. A. (2003) "*Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!*" (http://www.dwheeler.com/oss_fs_why.html)

compañía grande es de $2'322,000 dólares, para una compañía mediana es de $1'331,000 dólares y para una compañía pequeña es de $434,000 dólares. *Gran parte de estos proyectos fracasarán. Los proyectos de desarrollo de software son un caos.*" (cursivas mías).

La investigación del Grupo Standish [Standish 1995] muestra que el 31.1% de los proyectos serán cancelados antes de que se completen. El 52.7% de los proyectos costarán 189% más de sus estimaciones originales. El costo de estas fallas y sobrecostos es sólo la punta del iceberg, ya que los costos de oportunidad perdidos son de miles de millones de dólares. Por ejemplo, el fracaso de producir software confiable para manejar el equipaje en el aeropuerto de Denver le costó a la ciudad $1.1 millones de dólares por día [*idem*]. Y como este proyecto hay muchos casos, desgraciadamente pocos han sido documentados. [Standish 1995] cita que el 50% de los proyectos se consideraron operativos, pero *no* exitosos. El proyecto de software promedio se sobrepasa en su programación en la mitad o más. El 75 % de los productos de software grandes se entregaron a los clientes pero tienen fallas, son un fracaso porque no se usan o no cumplen los requerimientos del cliente. Los factores de daño o cancelación de los proyectos de software se muestran en la tabla 1.

Tabla 1. Factores de Falla o Cancelación en los Proyectos

Factores de Daño o cancelación	**%**
Requerimientos incompletos	13.1
Deficiencia en el involucramiento del usuario	12.4
Deficiencia de recursos	10.6
Expectativas no realistas	9.9
Deficiencia en soporte ejecutivo	9.3
Cambios en los requerimientos y especificaciones	8.7
Deficiencia en la planeación	8.1
Ya no se necesita más	7.5
Deficiencia en administración de TI	6.2
Desconocimiento en tecnología	4.3
Otros	9.9

La investigación sobre el fracaso de proyectos de software es bastante limitada y la mayoría se concentra en considerar los fracasos de manera genérica como "fallas en la administración de los proyectos". [Markus 2000] considera que "la mayoría de los fracasos pueden atribuirse a factores fuera de control del administrador del proyecto y su equipo". Las causas origen las identifica en "el sobrecompromiso del personal de ventas, el mal entendimiento de quienes eran los clientes reales, el conflicto entre las necesidades del cliente desconocidas por equipo de proyecto, falta de un staff técnicamente calificado, alta rotación del staff y la predominantemente percepción negativa que la mayoría de los usuarios tienen de su departamento de informática". También se apoya en las estadísticas del Standish Group. A esta recurrencia de fracasos se le ha dado en llamar la *Crisis del Software*, que por razones históricas desde fines de los años 60's se le sigue denominando de esa manera a pesar de que este fenómeno es más bien una aflicción o enfermedad cónica [Press 2002, p.8].

Para [McManus 2003], la fase preliminar de un proyecto debe incluir "un acuerdo sobre el nivel de servicio que incluya la justificación, una delineación sobre el enfoque global, la estimación de recursos (habilidades/personal, efectivo), un resumen de los riesgos y el enfoque para manejar los riesgos". En la generalidad de los proyectos de software, el patrocinador del proyecto se niega a pagar esta fase, entre otros argumentos por considerarlo como sencillo, de poco valor y que es la responsabilidad del desarrollador del proyecto. Lo que provoca grandes tasas de fracasos por la indefinición precisa de las necesidades o requerimientos. Esto se puede notar a partir del origen de los errores en el software que, según Walraet, citado por [Zavala 2003] es el siguiente:

Tabla 2. Origen de los Errores de Software

Fase	**%**
Estudio y análisis	56%
Diseño	10%
Código	7%
Otros	27%

En la estadística se nota que las fases donde se generan más errores en el software son las fases de estudio, análisis y diseño, que al hacerse con vaguedad e imprecisiones, provoca los problemas ya enunciados. Respecto al costo de corregir un error, según Walraet, citado por [Zavala 2003] es el siguiente:

Tabla 3. Costo Relativo de Corregir un Error de Software

Fase	**%**
Estudio y análisis	82%
Diseño	4%
Código	1%
Otros	13%

Se nota de manera inmediata que las etapas son consecuentes, aunque no se desarrollen completamente y que los errores en las primeras etapas impactan en mayor medida que las subsecuentes. Pero a pesar de esto, las organizaciones consideran a los proyectos de software como cualquier otro proyecto a pesar de que tienen un mayor impacto en los aspectos estructurales de operación de la organización.. El proyecto de software presenta una paradoja: al ser digital es más complejo, igual que muchos otros tipos de proyectos y el insumo y el desperdicio es fuerza de trabajo y en menor medida materiales. Por ejemplo, en los proyectos de producción como la fabricación de un nuevo producto, se destina gran parte a la investigación y desarrollo del producto que culmina con una serie de modelos, diseños y prototipos, antes de que esté listo para la producción en masa, que evidentemente, tienen un costo para la organización. En cambio, cuando es un proyecto de software cuyo entregable el un sistema de cómputo para la organización, que solo puede probarse plenamente cuando está terminado, la organización empuja directamente a la producción "sobre la marcha", con escasa rigurosidad en el diagnóstico de la propia organización y de sus necesidades de automatización. Los argumentos son, por lo regular, "la falta de tiempo" o que "los detalles se resolverán más delante" y escasas veces se cumple con las consecuencias de esta manera de proceder.

Muchos proyectos fracasan por no realizar el proyecto de factibilidad técnico-financiera y los beneficios son sobreestimados, los costos subestimados y los tiempos subestimados, la mayoría de las veces sin bases firmes para estimación.

La mayoría de las organizaciones consideran que los proyectos de software deben ser sencillos y baratos. Es frecuente que cuando el proyecto se le encarga a una empresa de consultoría, ésta se ve obligada a asumir el costo de ese diagnóstico, considerando que en la etapa de desarrollo "recuperará lo perdido". Sin embargo, dependiendo de la complejidad del proyecto puede salirse de control y tornarse en un proyecto fracasado por falta de un diagnóstico organizacional adecuado.

Para aclarar más esta problemática le pregunto, a Usted amigo, lector: "*¿Entraría Usted a cirugía en un quirófano sin un análisis previo?,"* Por supuesto que no! -seguramente contestará Usted. Esa es la misma pregunta que se hacen los ingenieros en software, cuando "en una hoja de papel se dibuja un bosquejo de una interfase de usuario" y se pide que a partir "de lo que se dijo en la junta se desarrolle el sistema". Y cuando se desarrolla una propuesta del proyecto "se viene abajo, sencillamente, porque el cliente, no acepta hacer y mucho menos pagar el diagnóstico de la operación de su propia organización" para el proyecto, ya que esto es intangible y como toda consultoría es costoso. La regla en la industria es que "el consultor debe ser lo suficientemente hábil como para detectar la problemática organizacional con la profundidad requerida con una *simple ojeada*", algo obviamente

imposible, o resignarse a *"perder el proyecto"*, por lo que no es sorprendente que fracase.

Según [Pinto 1990] "entre las causas de falla de los proyectos [de cualquier tipo] se encuentran cambios al medio ambiente en aspectos legales, sociales, políticos, tecnológicos y/o económicos que provocan que se retiren los fondos a los proyectos." Según el mismo autor, "la falla del proyecto está asociada a tres aspectos: (1) el proceso de implementación en sí, (2) el valor percibido del proyecto y (3) la satisfacción del cliente con los entregables del proyecto." Es evidente que algunas de las causas son estas, sin embargo, el autor, solo les atribuye el 40% a estos aspectos y exhorta a una investigación más profunda.

Uno de los grandes problemas de la industria del software es que a pesar de que hay estándares, metodologías, técnicas, lineamientos y demás herramientas, éstas no se emplean de manera generalizada, haciendo de esta industria algo menos que una artesanía. Además, los profesionales en software en su gran mayoría tienen deficiencias académicas importantes y muchos de ellos son generalistas (o todólogos) en vez de especialistas. Bajo los esquemas *ad hoc* que adopta la industria no se puede repetir ni predecir el proceso de producción, ni estimar la calidad del producto final. Afortunadamente se está dando una convergencia hacia la adopción de estándares industriales como Capabiliy Mature Model (CMM)[10].

[10] Ver el repositorio de investigaciones del Software Engineering Institute de la Carnegie Mellon University (http://www.sei.cmu.edu) financiado en gran medida por el Departamento de Defensa (DoD) de los E.U.

El aprendizaje organizacional

Según [McManus 2003], Ewsi-Mansan y Przasnski observaron que pocas organizaciones consideran como aprender de sus errores y en una encuesta descubrieron que el 60% de las organizaciones en los Estados Unidos terminaron más de un proyecto por las mismas razones. Además de que un desconcertante 75% de estas organizaciones fracasaron en conservar registros de sus proyectos fracasados. Lo que muestra la deficiencia de las organizaciones para capitalizar sus conocimientos.

No hay mucha investigación sobre las causas que provocan que las organizaciones "oculten" los proyectos fracasados, pero es fácil suponer que se debe en gran medida a que políticamente es un problema para los directores e involucrados en estos proyectos. [Bohem 2000] asegura que es peligroso asumir que los proyectos cancelados son fracasados porque induce a los administradores del proyecto de software a asumir que continuar el proyecto desperdiciará los recursos de la compañía y que probablemente cancelar el proyecto lo hará un gerente fracasado y afectará su carrera y que mejor no dice nada y esperar que pueda transferir el proyecto a otro nuevo. Bajo esta óptica es claro que nadie quiere asumir los costos de un fracaso profesional.

Para [Bohem 2000], "el *Chaos Report* del Standish Group etiqueta los proyectos cancelados como 'fracasados' e implica que el 31.1% de ellos se cancelaron por administración de software deficiente. Esta implicación es falsa y peligrosa. Es falsa particularmente en la era del cambio rápido, donde una cantidad de proyectos de software son iniciados adecuadamente, bien administrados y adecuadamente terminados antes de completarse porque sus supuestos originales han cambiado." Esto enfoque permite

resolver un poco el dilema de la aceptación de la cancelación de los proyectos de software, principalmente por haber cambiado el entorno. ¿Cómo aprender a desarrollar proyectos si no hay aprendizaje, si no se ejecuta lo que se ha dado en llamar la etapa *postmortem* del proyecto o se despide a los involucrados?

Para [McManus 2003], los síntomas de una *entrega* deficiente de los sistemas de información son los siguientes:

- Solicitudes de cambio frecuentes por usuarios
- Los usuarios tienen un deficiente entendimiento de sus propias necesidades
- Tareas no consideradas
- Comunicación insuficiente
- Deficiencia en una metodología adecuada y lineamientos para estimación
- Deficiencia de coordinación del desarrollo de sistemas
- Tiempo insuficiente para pruebas
- Deficiencia en la preparación
- Alineación de la estrategia de negocios deficiente

Estos síntomas listados son muy frecuentes en los proyectos de software y prácticamente están generalizados en las organizaciones de todo tipo y tamaño.

Los Factores de Éxito de los Proyectos

Los factores que afectan el éxito de los proyectos según Baker, Murphy y Fisher, citados por [McManus 2003], quienes estudiaron 650 proyectos en los Estados Unidos son los siguientes:

1. Compromiso con el proyecto en el establecimiento de calendarizaciones, presupuestos y objetivo de desempeño técnicos.
2. Frecuente retroalimentación de la organización patrocinadora
3. Frecuente de retroalimentación del cliente
4. Compromiso del cliente, del patrocinador, comprometido en el establecimiento de calendarizaciones, presupuestos y objetivo de desempeño técnicos.
5. Estructura de la organización adecuada al equipo del proyecto
6. Participación del equipo del proyecto en la determinación de la calendarización y los presupuestos
7. Entusiasmo del patrocinador
8. Deseo del patrocinador de crear las capacidades internas
9. Procedimiento de control adecuados, especialmente en relación con los cambios
10. Uso con juicio de las técnicas de programación en red
11. Un mínimo de agencias públicas y de gobierno involucradas
12. Falta de un gobierno excesivo
13. Soporte público entusiasta
14. Falta de impedimentos legales

Por su parte [Pinto 2000] propone utilizar los siguientes factores como críticos del éxito de un proyecto:

Tabla 4. Factores críticos de éxito de un proyecto de software

Factores críticos de éxito de un proyecto
Misión del proyecto – Metas y direcciones generales definidas con claridad al inicio del proyecto
Soporte Administrativo de Alto Nivel – Ayuda de la alta dirección para proveer los recursos necesarios y la autoridad y poder para el éxito del proyecto
Auscultación del Cliente – Comunicación, auscultación y escucha activa de todas las partes impactadas
Personal – Reclutamiento, selección y entrenamiento del personal necesario para el equipo del proyecto
Tareas Técnicas – Disponibilidad de la tecnología y experiencia (*expertise*) necesarias para el cumplimiento del las acciones técnicas específicas
Aceptación del Cliente – El acto de "vender" el final del proyecto a los usuarios finales
Monitoreo y Retroalimentación – Provisión a tiempo y de manera adecuada de información de control en cada una de las etapas del proceso de implementación
Comunicación – La provisión de una red apropiada y datos necesarios para todos los actores clave en la implementación del proyecto
Resolución de Problemas – Habilidad de manejar crisis inesperadas y desviaciones del plan

Como se nota en esta tabla, muchos de los factores de éxito de los proyectos de software están asociados a la administración del proyecto.

Gran parte de los enfoques que abordan la solución de la crisis del software básicamente se circunscriben a los siguientes enfoques:

1. El producto que se enfoca en mejorar el nivel de la calidad de los entregables del proyecto: modelos, documentos, código, etc. Tecnologías destacadas: *Unified Modeling Language* [Lenguaje de Modelado Unificado], análisis y diseño orientado a objetos, programación orientada a objetos, entre las más importantes.

2. El proceso de desarrollo mediante la adopción de modelos de ciclo de desarrollo y modelos de calidad que equivale a la *administración de proyectos* mediante el aprendizaje de técnicas de gestión por parte de de los administradores y administración de personal y el mejoramiento y predicibilidad de los resultados. Tecnologías destacadas: *Proceso Unificado*, *Capability Mature Model*, ciclo de vida evolutivo, administración de proyectos, entre otras.

3. El personal que se ha enfocado a desarrollar un modelo de equipo de trabajo y procurar las técnicas, metodologías y herramientas de desarrollo necesarias para manejar la complejidad del sistema a desarrollar. Tecnologías destacadas: *Team Software Process*, *Personal Software Process*, organización de equipo, liderazgo, motivación, etc.

Este enfoque en tres aspectos se le conoce como las tres P's de un enfoque de ingeniería de software, que considera la aplicación de metodologías y técnicas tradicionales y "precisas y rigurosas" de la ingeniería en el desarrollo de software.

Cada uno de estos enfoques ha tenido éxito parcialmente, pero ninguno ha reducido la complejidad, lo que Fred P. Brooks Jr. ha dado en llamar el mito de la "*No Silver Bullet*" en su célebre ensayo en 1986 (republicado en 1987): "No hay un solo desarrollo, tecnología o técnica de administración que por sí misma prometa mejorar aún en un orden de magnitud en la productividad en la confiabilidad, en simplicidad" [Brooks 1987]. Por otro lado, los proyectos de software presentan la paradoja que expresa la *Ley de Books* de su ensayo *The Mythical Man-Month*: "Muchos proyectos de software se han vuelto perversos más por falta de tiempo que por todas las otras causas combinadas", de modo que "al agregar fuerza de trabajo a un proyecto de software retrazado, lo retrasa más, de tal suerte que siempre se puede calendarizar el trabajo con pocos hombres y más meses, pero no siempre se pueden crear calendarizaciones con más hombres y menor meses" [Brooks 1995]. Luego entonces, se requiere un enfoque distinto, que se antoja un enfoque organizacional.

[Dhillon 2003] relata el caso de un proyecto de software en el *Departament of Motor Vehicles* (DMV) del estado de Nevada, E.U. El DMV era una de las unidades más grandes en el estado que empleaba a 2,200 personas en 36 oficinas en todo el estado. Las responsabilidades del departamento incluyen el reforzamiento de la ley, relacionar vehículos y trabajos de reparación de agua, expedición de licencias de vehículos. Además de la Comisión de Servicio Público, la División de Licencias de Conducir organizada en varias divisiones. Manejaba 131,000 registros de nuevos vehículos por mes, 6,000 licencias de conducir y 30,000 renovaciones por mes, cerca del 65% del sudeste de Nevada.

El proyecto del DMV, llamado Génesis trastocó todos los órdenes de poder en la organización intentando cambios en la estructura, los sistemas, la gente y la cultura. El costo del proyecto se estimó en $34 millones de dólares y con un presupuesto destinado final de cerca de $173 millones de dólares. Los tiempos de retraso del servicio de 40 minutos se fueron a 7 horas. Los cargos a los vehículos por clasificación incorrecta se fueron de $8 dólares a $100 dólares extra,

además de problemas con la expedición de licencias de conducir, todo por datos erróneos o faltantes. Los problemas se convirtieron en un desastre cuando una gran parte de la organización no pudo movilizar y delegar poder para realizar sus acciones de manera adecuada. En cambio, en aquellas áreas donde la estructura, los sistemas, la gente y la cultura se alinearon resultaron exitosas. Hubo una serie de falta de contratación de personal por malas estimaciones de los ahorros [*ídem*].

En el proyecto Génesis hubo problemas a nivel formal e informal en cómo se diseñó el sistema. Cuando el sistema falló, el gobernador del estado anunció un plan de emergencia de cinco puntos para aliviar la situación. Se contrató un staff temporal de 42 personas y a 24 horas de haber iniciado operaciones el proyecto entró en operación el sistema de renovación de licencias de conducir por correo. Entró en operación un programa de 30 días de gracia para renovación [*ídem*].

La experiencia de este proyecto sugiere que es más importante resolver el cambio organizacional que cualquier problema tecnológico de implementación. El poder organizacional es uno de las variables más importantes que deben *entenderse* apropiadamente y apalancarse antes de asegurar el éxito en la implementación de TI.

Las lecciones de este proyecto son:

1. Entender el poder, los procesos y los significados es un precursor de la implementación exitosa de un sistema de información.
2. Adicionalmente, entender las dimensiones del poder es importante para resolver varias interrogantes de alineación en relación con cambios en la estructura, los sistemas, la gente y la cultura.
3. Una consideración y entendimiento adecuado del poder creado en el sistema es esencial para cualquier implementación exitosa de un sistema de información.

Un par de proyectos fracasados

A continuación se muestra de manera general la forma en que se ejecutaron un par de proyectos de software en México[11].

En el proyecto, la compañía identificada como *WhatCo* es una empresa de recursos humanos con presencia nacional e internacional. Es una empresa altamente jerarquizada y especializada en Administración de Recursos Humanos. Tiene un área de sistemas con personal tanto interno como bajo *outsourcing*. En la dirección de sistemas se tienen deficiencias relacionadas con el personal, el proceso y el producto. Las herramientas de desarrollo impactan en la demora de la finalización del proyecto y comprometen la calidad del producto a medida que los costos se elevan.

En las personas involucradas para llevar a cabo la ejecución del proyecto se observa lo siguiente:

1. La mayoría carecen de estudios formales en computación o informática, por lo que carecen de conocimientos básicos sólidos. Los salarios son bajos y el nivel de motivación es bajo que se manifiesta en que *WhereCo* no provee de las facilidades para que su personal se mantenga (auto) actualizado y éste no realiza el esfuerzo por su cuenta para capacitarse. Por lo mismo, no se actualizan con las tendencias de la tecnología y conocimientos (ingeniería de software, base de datos y herramientas de modelado) y se vuelven

[11] Agradezco la aportación de los MC A. Ríos y H. A. Reyes sobre los datos de los proyectos.

"obsoletos intelectuales" y desconocen los mejores métodos para desarrollo del software. En pocas palabras, "están empolvados en el conocimiento" con deficientes bases de su profesión. Esto provoca deficiencia en el análisis, diseño, implementación y prueba del software.

2. La empresa no mantiene una política salarial competitiva y acorde al puesto para el personal de sistemas, además de que algunos miembros del equipo son *improvisados* en el área de sistemas debido a que con el fin de no despedirlos se les asigna a sistemas.

3. Existe un ambiente politizado donde se forman grupos informales que dividen y se mueven de acuerdo a sus propios intereses. "Se bloquea el trabajo con tal de no hacer cambios". No proporcionan información completa de su parte. "Sienten que saben más que el usuario final" que usará el software y que se considera el experto del dominio del problema y "lo ignoran no tomándolo en cuenta en la definición del producto".

4. Los directivos en lo más alto de la estructura no tienen contacto con el personal operativo (*"la tropa"*). "Los directivos solo piden que se entreguen las cosas en el tiempo acordado pero nunca los toman en cuenta, ni preguntan qué se necesita para salir a tiempo".

5. Los directores "no quieren que se conozcan sus procesos" por temor a sus competidores (internos) y ésta práctica se convierte en una práctica de los grupos informales. Algunos jefes de área quieren mantener su coto de poder y al mismo tiempo la empresa pierde capacidad de operación.

6. "El desarrollador es juez y parte". El mismo desarrollador analiza (brevemente), (se salta el diseño), codifica, prueba. La tasa de retrabajo es muy alta.

7. Al terminar el software después de 9 meses, el usuario final rechaza el software debido a que falla constantemente el módulo desarrollado y el desarrollador entra en un estado de crisis al perder su bono de desempeño. Algunas "cabezas" ruedan y otras se mantienen. Todos se culpan mutuamente, aunque no se diga abiertamente.

El proceso de desarrollo, es decir, la manera de cómo hacer las cosas en el área de sistemas también influye en la calidad del software:

1. La alta dirección solicita el desarrollo de un módulo de software "sin tomar en cuenta a los usuarios finales, que son los responsables de esa unidad de negocio de la empresa". Luego, "el desarrollador establece sus propias ideas de cómo debe ser el proceso de trabajo" que apoyará el software. Tampoco se involucra al ingeniero de proceso, que se supone tiene el conocimiento de los procesos de la organización. Lo curioso es que cuando está terminado el software se involucra al ingeniero de proceso para que "narre el proceso", cuando debería ser al principio.

2. Formalmente el software se debe apegar al proceso de negocio. Tanto el software y el documento del proceso de la unidad de negocio deben reflejar lo mismo.

3. No existe la formalización de los procedimientos de negocio. Por norma, debería tenerse documentada y actualizada la forma de cómo se hacen las cosas en cada unidad de negocio, siempre. "Nunca hay tiempo extra" y por lo mismo la dirección tampoco se preocupa por encomendar en plasmar los procedimientos de cada unidad de

negocio (facturación, clientes, nómina, etc.). Esto provoca demasiada ambigüedad en cuanto a la interpretación del proceso de trabajo. No hay forma de contrastar la validez de lo que se está modelando en software.

4. No existe la formalización de los procedimientos de desarrollo de software.

Por otro lado, las herramientas de los desarrolladores (software) son de modelado, construcción y pruebas del sistema.

1. Las herramientas de modelado que se tienen no son suficientes para las necesidades, por ejemplo, no se puede representar de manera abstracta el mundo real con técnicas de abstracción. Estas herramientas están muy precarias, por no decir que no hay, teniendo que modelar en papel, perdiendo la eficiencia productiva.

2. Las herramientas de programación son lentas para introducir código. Con editores de línea (modo carácter), lo que se convierte en una traba para el desarrollador. Son herramientas tecnológicamente atrasadas con 20 años, atadas a una plataforma tecnológicamente ya en proceso de obsolescencia.

3. Las herramientas para prueba automáticas. No hay programas que destruyan al software hecho por el desarrollador.

Conclusiones: Estructuralmente hay deficiencias organizacionales que están impactando en los recursos humanos, que tienen una motivación muy deficiente, además de la formación de grupos informales que luchan por el poder en varios niveles de la organización. Los procesos de producción son desconocidos a detalle por los niveles ejecutivos y los responsables de documentar los procesos. Los usuarios finales no son tomados en cuenta. La alta dirección no tiene "roce" con su niveles ejecutivos.

A continuación la forma en que se ejecuta un proyecto de software en la compañía identificada como *UniCo*. Es una organización pública de presencia nacional e internacional. Tiene una estructura altamente burocrática y un área de sistemas con personal tanto interno bajo nómina como bajo contrato, en su mayoría muy joven y recién egresados.

En la dirección de sistemas se tienen ventajas competitivas con el personal, el proceso y las herramientas que impactan en la calidad del proyecto y la calidad del producto.

Las personas involucradas para llevar a cabo la ejecución del proyecto se observa lo siguiente:

1. Cuentan con estudios formales en computación o informática, por lo que poseen conocimientos básicos sólidos. El nivel de motivación es alto. *UniCo* provee las facilidades para que su personal se mantenga actualizado, y éste se capacita por su cuenta. Se mantienen actualizados con las tendencias de la tecnología. En pocas palabras, cuentan con suficientes bases de su profesión.

2. El proyecto es ordenado por la alta dirección por un compromiso político con una organización externa.

3. El equipo de desarrollo del proyecto hace una evaluación del proyecto y le encuentra seria deficiencias, cuyas objeciones no son aceptadas por la alta dirección.

4. A medida que avanza el proyecto se complica y se excede el presupuesto original. Se tensan las relaciones entre los usuarios finales y el equipo de desarrollo y al interior del mismo equipo. La

rotación de personal se eleva y en un par de ocasiones la plantilla de personal prácticamente se sustituye por completo y terminan renunciando dos de sus líderes de proyecto. Para agilizar la ejecución los usuarios acuden a la alta dirección de *UniCo*.

5. El proyecto concluye después de 28 meses dejando atrás una ola de problemas y resentimiento entre el equipo del proyecto y los usuarios finales. El proyecto concluyó sin utilizarse. Lo que finalmente se desarrolló que fue una propuesta "muy simple y reducida" de lo originalmente se había estipulado. El proyecto se excedió en el presupuesto de unas 6.5 veces de lo planeado después de varias negociaciones legales.

6. Hoy nadie quiere saber siquiera del proyecto. Políticamente es riesgoso. "Es preferible que quede en el olvido".

Conclusiones: El equipo de desarrollo estaba muy motivado y al ser impuesto un proyecto con más expectativas que necesidades, el proyecto se sale de control para ambas organizaciones y concluye después de haber dejado tras de sí una ola de problemas a todos los niveles.

En ambos casos, el desarrollo del sistema no correspondía con las necesidades. No se involucró a los usuarios. El proyecto en ambos casos no se abordó como una estrategia sino como algo colateral. Los juegos políticos representaron los intereses de todos los involucrados a todos los niveles. Nunca hubo un diagnóstico de la organización para la cual se desarrollaría el proyecto.

El Enfoque Organizacional; Una Posible Salida a la Crisis del Software

Ante la problemática de la severa crisis del software y la pérdida de productividad, oportunidad y recursos, se propone abordarla de una manera distinta: utilizar un enfoque organizacional. A continuación se desarrollará este enfoque.

El principal supuesto de la propuesta es que la decisión de implantar un sistema de información en una organización debe ser una decisión estratégica, por lo que debe ser analizada con el debido cuidado y no dejarlo a la ligera. Bajo esta consideración debe hacerse un análisis estratégico con el escrutinio cuidadoso de los objetivos, las necesidades, los riesgos, las áreas, los procesos, las personas, los recursos y demás elementos de la organización que apoyarán el proceso de transformación. El sistema de información brinda la oportunidad para llevar a cabo una transformación organizacional mediante la reingeniería de sí misma, buscando el nivel más adecuado de automatización de los procesos administrativos y operativos y potenciar las ventajas competitivas transformando la cultura organizacional y no solamente subutilizar el potencial de las tecnologías de información concretándose a introducir algunos elementos de automatización, en lo esencial, operar "como se opera manualmente".

Los proyectos de software es que éstos pretenden modelar y apoyar la operación parcial de la organización. Es parcial ya que hay procesos como la toma de decisiones que prácticamente no se registran en ningún sistema de información. Por otro lado, toda organización presenta una dualidad; por un lado, la estructura formal y por otro la informal y el equipo de desarrollo del proyecto de software es incapaz de conocer la verdadera operación de la organización. Esta

es ***una causa estructural del fracaso de los proyectos de software***.

La *estructura formal* es aquella que formalmente está establecida mediante la visión, la misión, los estatutos, las actas de fundación, las leyes y reglamentos, las políticas, los procedimientos y los planes que organizan y orientan todos los recursos de la organización, asignando autoridad y recursos para cumplir sus fines. Esta estructura formal se denominará *"lo que debe ser"*. En cambio, la *estructura informal* es aquella estructura que se crea de manera paralela a la formal, que se construye en base a la estructura formal, pero que se deforma por las relaciones de autoridad, poder, subordinación, insubordinación, confianza y desconfianza bajo la autoridad carismática de los líderes. Esta estructura se denominará *"lo que es"*.

La organización opera por la combinación de ambas estructuras: cuando la estructura formal obstaculiza el funcionamiento, los miembros de la organización toman decisiones aún en contra de los procedimientos formalmente establecidos, otras veces porque los objetivos de la organización se contraponen a sus propios intereses. La estructura real que debería modelarse mediante software es la informal, *"lo que es"* realmente.

Cuando en el proceso de elaboración del software, el equipo de desarrollo, mediante sus analistas de sistemas intenta obtener los requerimientos del nuevo sistema de información, aborda la estructura formal de la organización y se enfrenta en el mejor de los casos con una situación atípica: obtiene de los usuarios finales y áreas de negocios, los informes, formatos, políticas, procedimientos y demás documentación, entrevista a los usuarios y en el menor tiempo posible, "obtiene" una serie de requisitos, vagos y la mayor de las veces contradictorios y poco realistas. La vaguedad se obtiene ya que en realidad los usuarios no realizan las operaciones de la organización tal como lo manifiestan, que es cercana a lo que los procedimientos estipulan que se debe hacer. En realidad, cada procedimiento finalmente se ejecuta de acuerdo al modo de realización del trabajador. Por ello, cuando se le requiere al usuario que especifique "cómo hace su trabajo", éste se encuentra ante la disyuntiva de "acatar la formalidad del procedimiento" o "decir que el procedimiento real es distinto". Frecuentemente opta por lo primero provoca que pocas veces "se descubra" la *verdadera* operación de la organización y que el sistema se desarrolle en base a requerimientos hipotéticos, al menos en lo que se refiere a los procesos de trabajo. La segunda opción provoca conflicto en el usuario, ya que al manifestar que está realizando el trabajo de una manera distinta a la especificada por el procedimiento puede ser objeto de una represalia de la organziación o le puede costar el empleo por su "falta". Este segundo escenario es poco común.

Hay otro elemento que provoca todavía más problemas en este proceso: el perfil de habilidades y personalidad del analista de sistemas. El analista es concebido como el profesional más bajo en el escalafón de puestos del equipo de desarrollo. Este analista se recluta normalmente con un perfil más orientado hacia los aspectos técnicos que hacia las habilidades de socialización, comunicación y relaciones humanas. Normalmente, el analista es un técnico con buenas capacidades en programación de computadoras pero escasas habilidades, sensibilidad y visión de la organización y que pocas veces logra establecer una buena relación de colaboración.

Esa relación de colaboración también se ve afectada por el hecho de que las áreas de producción de la organización consideran de manera frecuente a las áreas de informática o sistemas como "un mal

necesario", debido, entre otras cosas, a que no han asimilado el proceso de transformación al que se ve sujeta toda la organización, como mecanismo de defensa y de conservación del poder y/o debido a las malas experiencias con servicios informáticos como soporte técnico. Estos factores impiden que se logre una estrecha colaboración entre el analista de sistemas y el usuario.

Otro elemento importante es que los jefes inmediatos de los usuarios en realidad desconocen el detalle de los verdaderos procedimientos de la organización, sobretodo cuando se presenta una estructura rígida de autoridad y no hay suficiente confianza entre las líneas de mando. Por otro lado, la mayoría de los procedimientos escritos no están al día. Este escenario imposibilita que los operadores puedan expresar con efectividad la "realidad" de los procesos de trabajo de la organización. La mayoría de ellos expresan una abstracción hipotética de esos procesos, una mezcla entre "lo que se hace" y "lo que se debería hacer".

Otros muchos sistemas de información se inician a partir de los requerimientos que especifican los directivos en una junta en común acuerdo con el director del proyecto de software y la alta dirección.

Bajo los escenarios anteriores, ninguno de los procedimientos de obtención de los requerimientos del software próximo a desarrollar logra obtener la "verdadera" operación organizacional, aunque se aproxima. Esta es la razón de que cuando el sistema inicia las pruebas, el usuario rechace el sistema de información porque obviamente *no* representa la operación real y cambiar los procedimientos de manera unilateral es percibido con desagrado o malestar por el usuario. Esto implica que el usuario deba adaptarse al sistema, lo cual pocas veces ocurre o provoca más retraso en la operación, respecto al tiempo de operación antes de la introducción del sistema de información. Cuando se aceptan las pruebas a sabiendas de que el sistema de información no se adapta a la operación de la organización y no se detecta esa diferencia, se avanza a las siguientes etapas de desarrollo incrementando los riesgos de fracaso y los costos.

Otro aspecto importante a considerar es que cuando un sistema de información se introduce en una organización, éste altera la estructura de la organización, las relaciones de poder, los sistemas de operación, los procesos de producción, las relaciones laborales y por supuesto, la cultura. De todos éstos, la cultura organizacional es la más difícil de cambiar y la estructura laboral la que es más sensible. Cuando un sistema de información fracasa al entrar en operación puede paralizar el funcionamiento de la organización y puede llegar a afectar de manera importante los planes estratégicos y/o provocar conflictos laborales inesperados. La dirección debe comprometerse con el proyecto y apoyarlo decididamente. Debe iniciarse al mismo tiempo la sensibilización de todos los niveles de la organización y todos los miembros de la organización deben estar enterados del rumbo que tomará.

La teoría del la orientación a objetos (*object-oriented*) y el Lenguaje de Modelado Unificado (*Unified Modeling Language*) son herramientas conceptuales y tecnologías que permitirán la realización del modelado de las organizaciones con el nivel de abstracción, con el suficiente manejo de la complejidad y de manera gráfica.

La primera etapa de análisis debe ser de tipo estratégico y debe iniciar con un análisis organizacional y el análisis de las estrategias y necesidades de la organización con la gente más capaz y sensible que tenga la organización y el equipo de desarrollo de

software. El problema es determinar cómo potenciar la estrategia de la organización mediante su sistema de información. Posteriormente debe desarrollarse todo un proyecto que deberá administrarse para obtener resultados cercanos a los estimados. El modelado de sistemas será probablemente la etapa más creativa que se concretará cuando el sistema se despliegue en la organización y los objetivos de la organización se alcancen.

Conclusiones

1. El software es el activo más importante de las organizaciones y se ha convertido en el motor de la economía.

2. Los proyectos de software intentan modelar y automatizar parcialmente la operación de la organización y son tan complicados como lo es la propia organización.

3. Hay muchos factores que determinan el fracaso de los proyectos y otros tantos su éxito; conocerlos, es importante para incrementar el conocimiento organizacional, pero sorprendentemente, pocas organizaciones aprenden de sus errores.

4. Los proyectos de software deben ser emprendidos como parte del plan estratégico de la organización para aprovechar la oportunidad de transformar la organización y aprovechar las ventajas de implantar una automatización que le brinde ventajas competitivas y no solo "estar a la moda".

5. Para la obtención de requerimientos a partir de los usuarios se necesitan analistas con un perfil con habilidades de socialización, comunicación y relaciones humanas, con conocimientos tecnológicos y organizacionales con el fin de que detecte la dualidad de las organizaciones.

6. El factor más importante a considerar en un proyecto de software es que la organización es compleja por naturaleza y para modelar la organización es importante utilizar la teoría de sistemas, la teoría de orientación a objetos y la teoría de la organización, entre otras disciplinas para comprender mejor la operación de la organización con un enfoque multidisciplinario.

7. La capitalización de la experiencia en el desarrollo de software puede lograrse realizando la fase *postmortem* en los proyectos, documentando las experiencias, aprendiendo de los errores y entrenando a los miembros de la organización.

Literatura Citada

[Brooks 1987] Brooks, Fred (1987) "No Silver Bullet; Essence and Accidents of Software Engineering" *Computer Magazine*, reprinted april 1987. Disponible en http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html (2 mayo 2003)

[Brooks 1995] Brooks, Fred (1995) The Mythical Man-Month. Addison-Wesley, 2nd. Ed., Disponible en http://www.ics.uci.edu/~redmiles/ics121-FQ99/lecture/eleven/ (2 mayo 2003)

[Bohem 2000] Boehm, Barry (2000) "Project Termination Doesn't Equal Project Failure". *Computer*. September. pp. 94-96. Disponible en http://www.ieee.org (2 nov 03)

[Del Toro 2003] Del Toro, Jesús. 2003. Software: Vanguardia de la Nueva Economía, disponible en http://revistas.bancomext.gob.mx/Bancomext/rni/revista/octubre2003/PDF/ software.pdf (15 de nov 2003)

[Denning 2001] Denning, Peter. (2001) "The Profession of IT: Who Are We?". *Communications of*

ACM. February. (44):2. pp. 15-19. Disponible en http://portal.acm.org

[Dhillon 2003] Dhillon, Gurpreet (2003) "Dimensions of power and IS implementation" *Information & Management* (2003) disponible en línea en http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6VD0-49S80D2-1-3&_cdi=5968&_orig=search&_coverDate=10%2F14%2F2003 &_sk=999999999&view=c&wchp=dGLbVlb-zSkWW&_acct=C000048981 &_version=1&_userid=945819&md5=1dcea2ba2b18b86cdc87a2f83ab34042&ie=f.pdf (1 de noviembre 2003)

[Graham 2000] Graham, Stuart y Mowery, David C. (2000) "Intellectual Property Protection in the Software Industry". *National Research Council's conference on "Intellectual Property Rights"* Washington, D.C., Feb. 3, 2000. Disponible en: http://emlab.berkeley.edu/users/bhhall/swconf.doc (1 de noviembre 2003)

[IEEE 1990] IEEE. "Standard IEEE Std 610.12-1990 - Standard Glossary of Software Engineering Terminology", *The Institute of Electrical and Electronics Engineers*, Inc., NY. Disponible en http://www.ieee.org

[INCOSE 1998] INCOSE (1998). "What Is Systems Engineering?" International Council on Systems Engineering. Disponible en http://www.incose.org/whatis.html

[Intel 2003] Intel Corporation. "Expanding Moore´s Law". Disponible en: http://www.intel.com/labs/eml/index.htm (15 noviembre 2003)

[Markus 2000] Markus, Marcel (2000) "Failed software projects? Not anymore", *Quality Progress*; Nov; 33, 11; ABI/INFORM Global, pp. 116-117 Disponible en http://proquest.umi.com/pqdweb?index=63&did=000000064595761&SrchMode=1&sid=2&Fmt=6&VInst=PROD&VType=PQD&RQT=309&VName=PQD&TS=1067571760&clientId=39522 (1 de noviembre del 2003)

[McManus 2003] McManus, John y Wood-Harper, Trevor (2003) "Information systems project management: The price of failure", *Management Services;* May; 47, 5; ABI/INFORM Global, pp. 16-19 Disponible en http://proquest.umi.com/pqdweb?index=10&did=000000346162901&SrchMode=1&sid=2&Fmt=6&VInst=PROD&VType=PQD&RQT=309&VName=PQD&TS=1067570441&clientId=39522 (1 de noviembre del 2003)

[Panicc 2003] Paniccia, Mario y Borkar, Shekhar Y. (2002) "Silicon Photonics New Opportunities for Silicon" Intel. April 2002. Disponible en ftp://download.intel.com/labs/eml/ download/ EML_photonics.pdf (2 nov 2003)

[Pinto 1990] Pinto, Jeffrey K. y Mantel, Samuel J., Jr. (1990) "The Causes of Project Failure" *IEEE Transactions on Engineering Management*, (37):4, November. pp. 269-276 Disponible en http://ieeexplore.ieee.org/iel1/17/2268/00062322.pdf?isNumber=2268&prod=IEEE+JNL&arnumber=62322&arSt=269&ared=276&arAuthor=Pinto%2C+J.K.%3B+Mantel%2C+S.J.%2C+Jr.%3B (2 de noviembre del 2003)

[Press 2002] Pressman, Roger S. (2002) Ingeniería de Software; Un enfoque Práctico. McGraw-Hill Madrid. 5ª. Ed. 601 p.

[Standish 1995] Standish Group. “The Chaos Report”. Disponible en http://www.standishgroup.com/sample_research/chaos_1994_1.php (15nov03)

[Standish 1998] Standish Group. “The Chaos Report”. Disponible en http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf (15 de noviembre del 2003)

[Zakon 2003] Zakon, Robert H'obbes' "Hobbes' Internet Timeline v6.1" Disponible http://www.zakon.org/robert/internet/timeline/ (15/nov/03)

[Zavala 2003] Zavala Ruiz, Jesús. 2003. “Fundamentos de Sistemas de Información”. *Apuntes de Clase. Análisis y Diseño de Sistemas*. Fundación Arturo Rosenblueth. México, D.F. Disponible http://www.angelfire.com/scifi/jzavalar/far/ayds.html

[Zavala 2003b] Zavala Ruiz, Jesús. 2003. “La Crisis del Software”. *Apuntes de Clase. Análisis y Diseño de Sistemas*. Fundación Arturo Rosenblueth. México, D.F. Disponible en http://www.angelfire.com/scifi/jzavalar/far/ayds.html