Artículo **Revista de Cómputo Aplicado**
Marzo 2019 Vol.3 No.9 18-26

Diagnóstico de una organización de desarrollo de productos informáticos con perspectiva sistémica

Diagnosis of a computer products development organization with a systemic perspective

LEÓN-HERNÁNDEZ, Ciro David†, BADILLO-PIÑA, Isaías, GUTIÉRREZ-TORNÉS, Agustin Francisco* y CARRASCAL-ROMERO, Luisa Amalia

Escuela Superior de Ingeniería Mecánica y Eléctica, Instituto Politécnico Nacional
Escuela Superior de Ciencias y Tecnologías de la Información, Universidad Autónoma de Guerrero
Universidad Abierta y a Distancia de México

ID 1er Author: *Ciro David León Hernández*

ID 1er Coauthor: *Isaías Badillo Piña*, **CVU CONACYT ID**: 225165

ID 2do Coauthor: *Agustín Francisco, Gutiérrez-Tornés* / **ORC ID**: 0000-0002-8634-9152**, Researcher ID Thomson**: X-2283-2018, **CVU CONACYT ID**: 251621

ID 3rd Coauthor: *Luisa Amalia Carrascal Romero*

DOI: 10.35429/JCA.2019.9.3.18.26 Recibido Enero 10, 2019; Aceptado Marzo 30, 2019

Resumen

En este trabajo se realiza un diagnóstico con enfoque sistémico a una organización de Desarrollo de Productos Informáticos (DPI), utilizando la Cibernética Organizacional (CO) y el Modelo de Sistemas Viables (MSV) de Stafford Beer. El MSV proporciona adaptabilidad y capacidad de organización para disminuir la complejidad del entorno, aumentar la calidad del producto y evitar pérdidas de recursos, asimismo plantea cuestiones importantes y trae ideas relevantes para el desarrollo de software en situaciones que involucran la complejidad. El DPI puede estar apoyado de metodologías sistémicas para que los productos en las organizaciones sean de calidad para el usuario final, asegurando la viabilidad de la organización.

Cibernética Organizacional, Modelo de Sistemas Viables, Desarrollo de Software

Abstract

In this work, a diagnosis with a systemic approach is made to a Computing Products Development (DPI) organization, using Organizational Cybernetics (CO) and the Stafford Beer Model of Viable Systems (MSV). The MSV provides adaptability and organizational capacity to reduce the complexity of the environment, increase product quality and avoid loss of resources, also raises important issues and brings relevant ideas for software development in situations that involve complexity. The DPI can be supported by systemic methodologies so that the products in the organizations are of quality for the end user, ensuring the viability of the organization.

Organizational Cybernetics, Viable Systems Model, Software Development

Citación: LEÓN-HERNÁNDEZ, Ciro David, BADILLO-PIÑA, Isaías, GUTIÉRREZ-TORNÉS, Agustin Francisco* y CARRASCAL-ROMERO, Luisa Amalia. Diagnóstico de una organización de desarrollo de productos informáticos con perspectiva sistémica. Revista de Cómputo Aplicado. 2019, 3-9: 18-26

* Correspondencia al Autor (Correo electrónico: afgutierreztt@uagro.mx)
† Investigador contribuyendo como primer Autor.

Introduction

En la actualidad el software forma parte de la vida diaria de los seres humanos, podemos verlo a simple vista en los teléfonos inteligentes, tabletas, laptops, computadoras de escritorio, autos, hogar, comunicaciones, educación, medicina, etc., prácticamente en todas partes. Es importante conocer los factores que emplean las organizaciones en el DPI, como lo son: metodologías, modelos, herramientas, capital humano, infraestructura, dirección, comunicación, relaciones, conocimiento, etc.

Los sistemas informáticos (SI) se desarrollan de manera particular para un cliente específico o bien para un mercado general. La manera de desarrollo varía dependiendo de la complejidad de la organización, entre más grande es el requerimiento, más complejo es el sistema a desarrollar.

La informática aporta herramientas y procedimientos, que con el apoyo de la Ingeniería de Software mejora la calidad de los productos, aumenta la productividad, facilita el trabajo a los desarrolladores, el control de los procesos y suministra las bases para construir software de alta calidad en una forma eficiente, Gacitúa (2003).

Para solucionar este problema se utiliza la Ingeniería de Software (IS)[3], ya que apoya el desarrollo en la organización y es necesaria para aplicar un enfoque sistemático, disciplinado y cuantificable al ciclo de vida del software, que está conformado por el análisis, diseño, programación, pruebas, implementación, mantenimiento y obsolescencia.

La Cibernética Organizacional (CO) es uno de los enfoques sistémicos que, derivado de la Cibernética creada por Wiener (1948), aplica los principios relacionados con la Comunicación y el Control propios de la Cibernética a las Organizaciones. Este desarrollo teórico y metodológico ha sido realizado por Stafford Beer (Beer, 1979, 1981, 1985).

En este trabajo se realiza un diagnóstico de una organización de desarrollo de sistemas informáticos, apoyado en la CO y el Modelo de Sistemas Viables (MSV) de Stafford Beer.

Se considera que este enfoque pueda ayudar a diagnosticar y mejorar la estructura, procesos y actividades en una organización, con el fin de aumentar el nivel de casos de éxito, la productividad, la calidad en los sistemas desarrollados y la adaptación de la organización ante los cambios dinámicos.

La complejidad organizacional

"Los sistemas complejos se caracterizan por un gran número de componentes heterogéneos con un alto grado de interconexiones, relaciones y dependencias no lineales. Existen en un entorno de cambio dinámico que exige responder dinámicamente el comportamiento" (Arenque y Kaplan, 2000a). La heterogeneidad[4] resulta, en parte, por el esfuerzo de adaptación y evolución del sistema a su entorno, que cambia dinámicamente, para seguir siendo viable. Y cuanto más complejo sea el sistema, más difícil es poner en orden las acciones efectuadas con anterioridad. El equilibrio entre el ímpetu de evolución que impulsa al sistema hacia el futuro y la inercia de la herencia es una de las causas que explican la heterogeneidad de estos sistemas de software. Esa heterogeneidad constituye una parte inevitable de su complejidad.

Entre los factores que explican la necesidad de evolución y adaptación, destacan las mejoras de productos para aprovechar el progreso tecnológico, la aparición de nuevos mercados u oportunidades de negocio, la aparición de nuevas técnicas de desarrollo y, el aprendizaje de los usuarios finales. Estos factores son particularmente importantes en el caso del software que se encuentra en el borde de la tecnología, es decir en continua evolución.

La IS es de gran apoyo para el desarrollo y siempre debe ser practicada en una forma de reducir la complejidad percibida o real del sistema informático al mínimo. Esto significa el cumplimiento de los requisitos de software al hacer creer a sus usuarios "la ilusión de la simplicidad" (Booch, 1994). Esto sólo hace el desafío más grande para los desarrolladores, que, en última instancia, tienen que hacer frente a toda la diferencia entre la complejidad implícita en las especificaciones de requisitos y la complejidad residual de la simplicidad, que los usuarios finales perciben.

[3] Aplicación de un enfoque holístico, disciplinado y cuantificable al desarrollo, operación, mantenimiento y mejora del software, además del estudio de estos enfoques.

[4] Que está formado por elementos de distinta clase o naturaleza.

ISSN-2531-2952

Esos factores que contribuyen a aumentar la heterogeneidad no son mutuamente excluyentes, de hecho, hoy en día todos ellos pueden ocurrir simultáneamente. Por ello la organización como un todo, tanto el proceso como la arquitectura de software, deben tener una gran cantidad de flexibilidad con el fin de auto organizarse y adaptarse a las condiciones cambiantes continuamente, esto incluye a las personas que gobiernan la organización. Las organizaciones necesitan ser más susceptibles de aceptar nuevos paradigmas para la gestión de la complejidad.

La complejidad de los sistemas, de los procesos y de la organización es de hecho inevitable, por lo tanto, será necesario discernir la naturaleza de esa complejidad. Es pertinente buscar un enfoque que sea capaz de aceptar y hacer frente a la complejidad en lugar de simplemente ignorarlo, o tratar de evitarlo.

La complejidad de los sistemas de software se desarrolla con el tiempo, está íntimamente conectada con la idea de la evolución, y el tiempo es un factor esencial en la génesis de sistemas complejos. Gall (1986) establece el principio según el cual "Un sistema complejo que trabaja es invariablemente identificado como una etapa de evolución a partir de un sistema simple que trabajó". "Un sistema complejo diseñado desde cero no funciona y no puede ser modificado para hacer que funcione". Tiene que empezar de nuevo, comenzando con la etapa inicial de diseño de sistema.

El MSV y el desarrollo de software

En el ámbito del Pensamiento Sistémico se encuentra la Cibernética Organizacional (CO) y el Modelo de Sistemas Viables (MSV) de Stafford Beer. Este modelo es conceptualmente consistente con los principios de la Cibernética y de la Administración de Sistemas de Actividad Humana. Es considerado como la cibernética de gestión, es decir, una aplicación de los conceptos y principios cibernéticos para la gestión y administración de las organizaciones. Con el MSV se identifican los fundamentos profundos de la viabilidad a través de un diagnóstico o diseño organizacional.

Si el MSV lo utilizamos para describir a una organización de desarrollo de software, veremos primero todo el ambiente en el que se desarrolla, el Sistema Operativo en donde podemos encontrar los diferentes perfiles que requiere el área como lo son; analistas, diseñadores, programadores, testers, etc. y por otro lado el Sistema de Gestión, quien se encarga de administrar los recursos de la organización. Este modelo se basa en lo que Beer ha llamado el teorema de "Sistemas Recursivos" que dice: "En una organización de estructura recursiva, cualquier sistema viable contiene y está contenido en otro sistema viable". A todo lo anterior Beer lo denomina Sistema 1, ver figura 1.

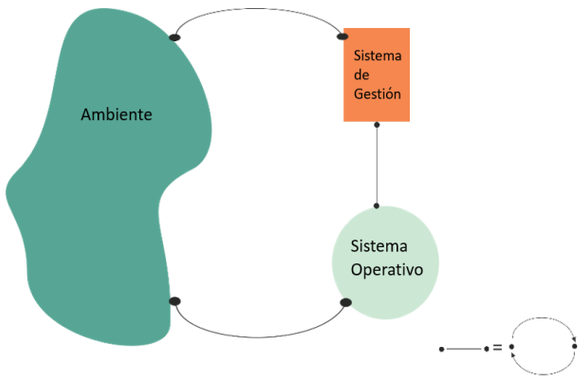

Figura 1 Teoría de los Sistemas Viables (Sistema 1)
Fuente: Elaboración Propia

Las organizaciones que se esfuerzan por desarrollar software en tales condiciones se vuelven complejas así mismas. Esto es una consecuencia natural de la Ley de la Variedad[5] Requerida (LVR) declarado por Ross Ashby: "únicamente la variedad puede destruir a la variedad" (Ashby, 1956). Ashby pretende mostrar, que una situación sólo pudo ser controlada si la variedad del controlador coincide con la variedad de la situación que se desea controlar, a mayor variedad dentro de una organización, mayor será su capacidad para hacer frente a la variedad en su Sistema Operativo, su entorno y todavía mantener su regulación interna, ver figura 2.

Otra consecuencia natural de la LVR, es que el proceso de desarrollo de software debe tener suficiente variedad para mantener el desarrollo de un sistema de software complejo bajo control.

[5] Variedad, en el lenguaje de la cibernética, es una de las medidas de la complejidad. Es el número de estados que el sistema en su conjunto puede presentar.

ISSN-2531-2952

Según Humphrey (1995), el proceso de un sistema es la secuencia de pasos necesarios para desarrollarlo o mantenerlo. El proceso de software establece el marco técnico y de gestión para la aplicación de métodos, herramientas y personas al desarrollo, mientras que la definición del proceso identifica los roles y especifica las tareas.

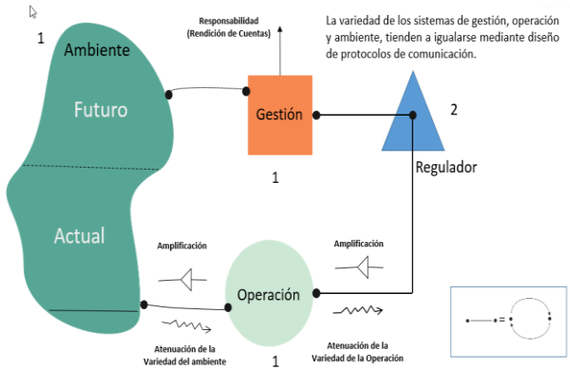

Figura 2 Ley de la Variedad Requerida o Ley de Ashby
Fuente: Elaboración Propia

Boehm (1988) explica que las funciones principales de un modelo de proceso de software son para determinar el orden de las etapas implicadas en el desarrollo y evolución de los sistemas informáticos y establecer los criterios de transición para avanzar de una etapa a la siguiente.

El proceso de software se encuentra entre la organización y el sistema que se desarrolla. Una vez que se dieron cuenta de la complejidad de ambos, ya que únicamente la variedad puede destruir a la variedad, entonces no hay esperanza para que el proceso de software puede ser simple. Sólo un proceso de desarrollo que al menos sea tan complejo como el software puede primero crearlo, y luego transformarlo para adaptarlo a las etapas de un ciclo de vida, ver figura 3, y en última instancia, generar su campo de posibilidades y mejorando radicalmente o hacerlo evolucionar.

El proceso de software es el medio a través del cual el sistema de software evoluciona progresivamente desde un sistema simple que funciona dentro de uno complejo. Se debe tomar en cuenta la capacidad de adaptación del sistema de software para coevolucionar contínuamente con su ambiente. El proceso de software debe exhibir capacidad de autoorganización para enfrentar la turbulencia del ambiente externo.

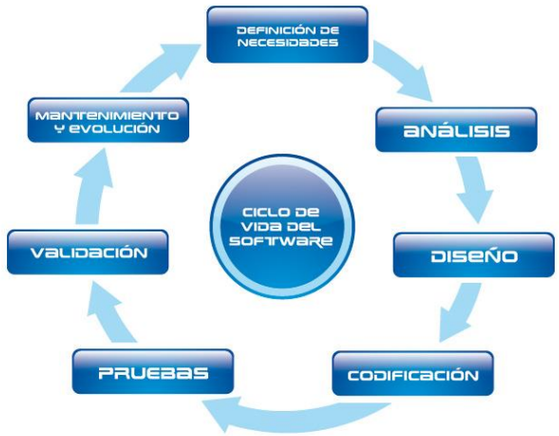

Figura 3 Ciclo de vida del desarrollo de software
Fuente: Elaboración Propia

El MSV, consta de cinco subsistemas básicos que son necesarios para garantizar la viabilidad total del sistema, es decir, su capacidad para mantener una existencia separada en un entorno en cambio continuo. Beer identifica los cinco subsistemas por números.

El Sistema Uno es el sistema de implementación, comprende las unidades autónomas de operación, las cuales tiene interacción con el medio externo. Beer (1979) denomina este Sistema como la Unidad Organizativa Elemental (UOE), se compone de una unidad de operación y su respectiva unidad de gestión en la interacción con su entorno ver Figura 1. La UOE produce el sistema, es decir, ejecuta las actividades que son esenciales para lograr el propósito del sistema.

Caso de estudio

Para aplicar la teoría mencionada anteriormente, se tomó como caso de estudio en la Ciudad de México al Centro Nacional de Cálculo (CENAC), el cual es un Órgano de Apoyo al Instituto Politécnico Nacional (IPN) en el desarrollo de sistemas informáticos para las necesidades de las diferentes áreas del IPN, en la automatización de procesos

Actualmente el CENAC está conformado por 70 personas, entre los cuales se encuentran 31 desarrolladores, 10 analistas, 10 testers, 10 administrativos, 6 jefes de departamento, 2 jefes de División y 1 Director.

ISSN-2531-2952

Se realizó el diagnóstico con el MSV, en donde los Sistemas Uno, se definen por su propio propósito, el cual contribuye al propósito de todo el sistema viable, en este caso las UOE´s típicas de desarrollo de software son: 1A-Ingeniería de Requerimientos, 1B-Ingeniería de Análisis, 1C-Ingeniería de Diseño, 1D-Ingeniería de Programación, etc.

Beer observa que varias unidades organizacionales aisladas, no hacen un sistema. Es necesario un metasistema[6] para explotar la sinergia y mantener la cohesión entre las unidades autónomas. El metasistema debe imponer las restricciones necesarias mínimas con el fin de no privar de la libertad las unidades del Sistema Uno. La libertad deja espacio para la creatividad, el cual es un requisito esencial para el esfuerzo de resolver problemas complejos y, en consecuencia, para la viabilidad.

Si la libertad del Sistema Uno llega a expandirse sin restricciones, decisiones descoordinadas y acciones entre las unidades podrían causar inestabilidad en el comportamiento dinámico de todo el sistema. Esas inestabilidades podrían terminar sobrecargando el sistema con un tipo de re-trabajo que podría ser evitado por la coordinación de las actividades simples. Un aparato anti-oscilatorio es entonces necesario.

Esta es la función del Sistema Dos, el sistema de control regulador, el cual es responsable de la coordinación entre el Sistema Uno autónomo y las otras unidades organizacionales. El Sistema Tres es el sistema de auditoría y monitoreo operacional. Es un sistema táctico a cargo de la parte interior y el ahora, vigila y controla las actividades internas e inmediatas, el mantenimiento de la homeóstasis[7] (equilibrio interno) y por lo tanto la viabilidad a corto plazo de todo el sistema. Incluye al Sistema Tres Asterisco, que es responsable de las auditorías esporádicas del metasistema directamente en las operaciones. Lo anterior es necesario porque los actores del metasistema no tienen un control completo sobre toda la variedad proliferada a través las unidades de operación autónomas.

Mediante auditorías y monitoreo, el Sistema Tres promueve las políticas y refuerza la adhesión a las normas, limitando así la proliferación arbitraria de la variedad.

El Sistema Cuatro es el sistema de inteligencia estratégica. Se trata de un sistema de control adaptativo que hace la integración con el entorno externo y ofrece un anticipo del futuro estado de los SI desarrollados y por desarrollar, así como del número de los usuarios finales, competencia, etc. Es un sistema estratégico que se refiere a la viabilidad a largo plazo de todo el sistema.

El Sistema Cinco es el sistema político. Se trata de un sistema de control de supervisión, que establece el propósito y los valores de la organización. Se establece el equilibrio entre Sistema Tres asuntos a corto plazo y el Sistema Cuatro asuntos a largo plazo, mediante un bucle homeostático, por ejemplo.

Las líneas de conexión entre los subsistemas en el MSV representan canales de comunicación bidireccionales. Estos canales son bucles de intercambio de variedad dinámica que opera continuamente a fin de mantener la estabilidad del sistema. Estos bucles se denominan homeostatos, después del término homeostasis, que en biología designa el equilibrio interno de los organismos vivos. Una aplicación meticulosa del MSV sugiere un análisis exhaustivo y cuidadoso diseño de cada uno de los canales de comunicación y sus transductores cada vez que cruzan las fronteras de los 5 subsistemas del MSV y del ambiente.

Un concepto clave en el modelo, es que la viabilidad es recursiva: para que el sistema sea viable, cada uno de sus subsistemas debe haber un sistema en sí viable. Dado que todos los cinco subsistemas son necesarios para la viabilidad, la función del Sistema Uno es reproducir en cada subsistema la misma estructura de todo el sistema viable con sus cinco subsistemas. Por lo tanto, la estructura recursiva del modelo refleja la naturaleza recursiva del concepto de viabilidad.

[6] Es todo aquello que se encuentra fuera de la frontera del sistema bajo estudio. Se denomina también entorno o medio ambiente.

[7] Conjunto de fenómenos de autorregulación, conducentes al mantenimiento de una relativa constancia en la composición y las propiedades del medio interno de un organismo.

ISSN-2531-2952

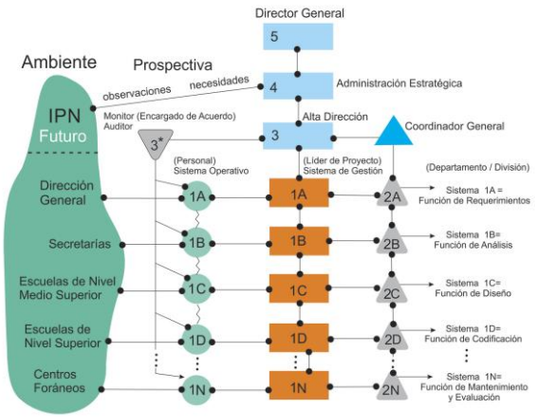

Figura 4 MSV de una organización de desarrollo de software
Fuente: Elaboración Propia

El modelo Sistémico - Cibernético de procesos y arquitectura de software

La programación en el complejo requiere del proceso de software y la arquitectura del sistema para ser capaz de responder a un entorno dinámico que impone cambios imprevistos. El MSV es un modelo de organización que ofrece esa capacidad. Así que en este artículo se propone el diagnóstico de MSV como un modelo de proceso de software.

El MSV es un modelo conceptual de la organización basada en principios cibernéticos. Es independiente del tipo de aplicación. Como tal, se puede utilizar para describir tanto la organización de la arquitectura de software (un patrón de diseño o patrón arquitectónico) y la organización de procesos de software (un modelo de proceso de software). A pesar de que la comprensión del MSV como un modelo de proceso de software podría ser muy sencillo en muchos detalles, unos pocos aspectos de aplicación que exigen la revisión de algunos conceptos tradicionales, ya que aporta nuevas perspectivas para la programación en lo complejo.

Una descripción detallada de la aplicación del MSV como modelo de proceso sería largo. Por esa razón, en este trabajo se centra en unos pocos puntos de vista mientras se revisan algunos conceptos tradicionales. Se omiten muchos detalles acerca de la interpretación del MSV como una arquitectura de software y el modelo de proceso como un análisis exhaustivo de los diversos canales de comunicación.

Si bien esos detalles están fuera del alcance de este artículo, esas omisiones no disminuyen el valor de las ideas que aquí se presentan, ya que esas ideas son el resultado de los principios básicos de la cibernética implicados en el MSV.

La modularización impulsada por los criterios de viabilidad

El MSV se construye a partir del Sistema Uno, la unidad organizativa elemental que produce el sistema. En este trabajo, la unidad organizativa elemental tiene una interpretación y aplicación de dos tipos. Por un lado, representa una unidad organizativa arquitectura del sistema de software, es decir, un componente de la arquitectura o módulo (del código). Por otro lado, representa una unidad de organización de procesos de software (de personas), que se encargan de la concepción, elaboración, construcción, adaptación y evolución de su respectivo módulo de arquitectura.

El criterio para identificar las unidades elementales es la viabilidad. Las unidades que serán entendidas como unidades elementales de la arquitectura de software son aquellos módulos que merecen una existencia separada. Entre las razones que justifican una existencia separada de un módulo, se puede mencionar que: (1) que incorpora alta tecnología y las necesidades de actualización constante con el fin de mantenerse a la vanguardia de la innovación; (2) que cumple con las necesidades de un mercado dinámico y tiene que seguir su evolución y (3) debe adaptarse de forma dinámica con el fin de satisfacer las necesidades cambiantes de sus usuarios y de cooperación con los otros módulos que son afines.

Cuando se hace referencia a la arquitectura de sistemas de software, la capacidad de adaptarse a los cambios imprevistos a veces se llama resistencia al cambio (Booch, 1994, 1996). Por lo tanto, un proceso de desarrollo de software que emplea el modelo cibersistémico fomente la resistencia del SI.

ISSN-2531-2952

Una arquitectura ideal para impulsar la evolución del sistema

En general, la tecnología evoluciona a un ritmo más rápido que el sistema de software puede seguir. Actores del sistema que presenten nuevos requisitos a un ritmo mayor que el sistema de software puede incorporar.

La lista de los requisitos pendientes de un sistema de software complejo tiende a crecer en el tiempo en lugar de la contracción. Debido a esto, el Sistema Cuatro tiene que ver con la viabilidad a largo plazo, puede mantener una imagen de la futura arquitectura basada en los requisitos pendientes, las tendencias tecnológicas y el aprendizaje con la arquitectura actual.

Por un lado, el Sistema de Tres mantiene la arquitectura conceptual que sirve como la base para la implementación del sistema de software actual. Por otra parte, el Sistema Cuatro anticipa la futura arquitectura. La futura arquitectura no está diseñada para su aplicación inmediata como en la arquitectura conceptual. Es un modelo ideal destinado a corregir los problemas de la arquitectura actual, dar cabida a mejoras a largo plazo y proporcionar capacidad de resistencia a los cambios futuros.

En el caso de los sistemas de software complejos, la implementación inmediata de las soluciones incorporadas en la futura arquitectura sería muy traumático, ya que el impacto de todos los cambios deseados capturados en la futura arquitectura implica una discontinuidad en el proceso de desarrollo. Esa discontinuidad podría significar una larga interrupción, sin embargo, el modelo ideal de la futura arquitectura sirve de referencia que guía la refactorización oportunista de la arquitectura conceptual.

Una vez que la arquitectura conceptual se actualiza y ya que guía la implementación, los cambios se transfieren al sistema concreto, ya que se traducen en soluciones de diseño y luego en el código fuente.

El concepto de refactorización se introdujo en el contexto de la programación extrema y otras metodologías ágiles como una parte integral del proceso de software (Fowler, 1999). El término 'oportunista' sugiere el análisis de riesgos.

Este argumento refuerza la adopción de un modelo de proceso impulsado por riesgo, tales como el modelo espiral, en el Sistema Tres. El Sistema de Tres retroalimenta, al Sistema Cuatro, cualquier problema de resiliencia puede identificar en la arquitectura conceptual actual, por lo que el Sistema Cuatro puede resolverlos en la imagen ideal de la futura arquitectura que se desarrolla. Además, el Sistema Cuatro introduce nuevos elementos en la futura arquitectura que hacen provisiones para los avances y las tendencias futuras. El Sistema Tres oportunamente incorpora elementos de la futura arquitectura de la arquitectura conceptual actual. Entonces, el Sistema Uno se convierte la arquitectura conceptual en la implementación concreta.

El papel del arquitecto es proporcionar un sentido de integridad conceptual de la arquitectura del sistema de software (Booch, 1996). El MSV asigna una nueva responsabilidad para el papel, es decir, el equilibrio de las preocupaciones actuales y futuras, a fin de promover la innovación sin tener en cuenta el valor de la herencia. Ese equilibrio asegura la viabilidad del proceso de software y la arquitectura de software.

Conclusiones

Desde que Ross Ashby enunció la Ley de la Variedad Requerida, no dejó esperanza para que la vida pudiera ser simple en un mundo complejo, una vez que se hace conocido que 'únicamente la variedad absorbe la variedad'. A veces podemos quejamos que los resultados de la complejidad del sistema de software de un proceso de software complejo, como si esa realidad pudiera ser diferente.

Sin embargo, la complejidad del proceso a menudo resulta de la complejidad de la organización, que es determinado por una combinación compleja de presiones externas del ambiente sobre las que la organización tiene poco o ningún control. De nuevo, esto no sirve como excusa para justificar complicaciones innecesarias que resultan de la decisión arbitraria de las mentes caprichosas.

Sin embargo, hay que ser conscientes del hecho de que existe complejidad en el medio ambiente y que explica gran parte de la complejidad que uno tiene que hacer frente en el proceso de software.

ISSN-2531-2952

Por lo tanto, no tiene sentido tratar de ocultarlo, fingiendo que no existe y con la esperanza de que podríamos tener un proceso sencillo software para resolver problemas complejos de desarrollo. En ese caso, esta ideal de simplicidad no es cibernéticamente consistente.

Es mejor entonces asumir que la complejidad de los sistemas de software y el proceso de software es el tiempo natural e inevitable, y aceptar el hecho de que uno es "programación en lo complejo". Sin embargo, la aceptación de este hecho no significa sucumbir ante la complejidad y rendirse a sus consecuencias implacables. Significa buscar soluciones alternativas que hacen que la programación en el complejo sea viable.

Se propone una composición del MSV de Beer y el modelo en espiral de Boehm en este documento como un enfoque alternativo. Se propone como un modelo de proceso de software para hacer frente a los retos de la programación en lo complejo. Programación en lo complejo se asocia con el desarrollo de sistemas de software en un entorno dinámico. Exige del proceso de software, la capacidad de autoorganización, a pesar de las condiciones inesperadas.

Además, se exige de la arquitectura de software, la capacidad de adaptación continua a los cambios imprevistos. El MSV admite mucha más complejidad que los modelos mecánicos utilizados convencionalmente en Ingeniería de Sistemas y análisis. Ayuda a la organización en un marco comprensible una gran parte de la complejidad que es inherente en el mundo real del desarrollo de software. Por otra parte, se trata de la cuestión de la viabilidad basada, como es, en las capacidades del sistema de auto organización y adaptación a los cambios.

Sin embargo, la aplicación de la MSV propuesto aquí no está destinado a abordar la idiosincrasia de la realidad social de la organización, que puede incluir cuestiones culturales y políticas pertinentes. Hay otras metodologías de sistemas que se pueden utilizar en combinación con el MSV para ese propósito. Por ejemplo, la Metodología de Sistemas Suaves y los Sistemas Críticos Heurísticos (Flood y Jackson, 1991; Jackson, 1991). No obstante, aborda los aspectos dinámicos del proceso de software.

Esto es particularmente útil, ya que a menudo es deseable incorporar algunas repetitividades en un proceso complejo con el fin de trabajar en la mejora de procesos, por ejemplo. La repetitividad es una ventaja típica de los enfoques mecánicos clásicos. Para comprender el proceso de software como un sistema viable añade la posibilidad de explorar su capacidad de adaptación a los cambios y para la auto-organización en respuesta a las perturbaciones del entorno. El argumento es que para la viabilidad del software (y la organización) no sólo el proceso de software (incluyendo la arquitectura, el diseño y puesta en práctica) debe ser entendida como un proceso dinámico. La propia estructura de la organización es dinámica y los sistemas de hardware y software debe corresponder a la estructura organizativa del MSV.

Referencias

Ashby WR. 1956. Introduction to Cybernetics. Wiley: London.

Bass LJ, Klein M, Bachmann F. 2001. Quality attribute design primitives and the attribute driven design method. Revised papers from the 4th International Workshop on Software Product-Family Engineering, 169–186.

Beck K. 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley: Boston.

Beer S. 1972. Brain of the Firm. Allen Lane: London.

Beer S. 1979. The Heart of Enterprise. Wiley: Chichester.

Beer S. 1985. Diagnosing the System for Organizations. Wiley: Chichester.

Boehm B. 1988. A spiral model of software development and enhancement. IEEE Computer 21(5): 61–72.

Boehm B. 2000. Spiral Development: Experience, Principles, and Refinements, Spiral Development Workshop, Special Report CMU/SEI-2000-SR-008. CMU/SEI: Pittsburgh.

Booch G. 1994. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings: California.

ISSN-2531-2952

Booch G. 1996. Object Solutions: Managing the Object-Oriented Project. Addison-Wesley: California.

Deremer F, Kron H. 1976. Programming-in-the-large-versus-programming-in-the-small. IEEE Transactions on Software Engineering SE-2, 2: 321–327.

Flood RL, Jackson MC. 1991. Creative Problem Solving:Total Systems Intervention. Wiley: Chichester.

Fowler M. 1999. Refactoring. Improving the Design of Existing Code. Addison-Wesley Professional: Boston, MA.

Gall J. 1986. Systemantics: How Systems Really Work and How They Fail (2nd ed.). The General Systemantics Press: Ann Arbor, MI.

Herring C. 2001a. The pattern of the viable system and its language. KoalaPLoP, Melbourne, Australia.

Herring C. 2001b. Adaptable and adaptive systems: the intelligent control paradigm for software architecture. Working Conference on Complex and Dynamic Systems Architectures, Brisbane, Australia, 12–14.

Herring C, Kaplan S. 1999. Viable components: a cybernetic organization for computing, International Conference on Technology of Object-Oriented Languages and Systems. TOOLS Pacific, 2nd Australian Workshop on Software Architectures, Melbourne, Australia.

Herring C, Kaplan S. 2000a. The viable system architecture. Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34), Maui, Hawaii.

Herring C, Kaplan S. 2000b. Viable systems: the control paradigm for software architecture revisited. Australian Software Engineering Conference, Canberra.

Herring C, Kaplan S. 2000c. The viable system model for software. 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000), Orlando, Florida.

Humphrey WS. 1995. A Discipline for Software Engineering. Addison-Wesley: USA. Jackson MC. 1991. Systems Methodology for the Management Sciences. Plenum: New York.

Kruchten P. 2000. The Rational Unified Process: An Introduction (2nd ed.). Addison-Wesley—Longman: Reading, MA.

Malan R, Bredemeyer D. 2002. Software Architecture: Central Concerns, Key Decisions. Bredemeyer Consulting: Bloomington, IN.

Wiener N. 1948. Cybernetics, or Control and Communication in the Animal and the Machine. The Technology Press/John Wiley & Sons, Inc.: Cambridge, MA/New York.

Use APA system. Should not be numbered, nor with bullets, however if necessary numbering will be because reference or mention is made somewhere in the Article.