Ingeniería del Software I

Guía de Estudio
Ciclos de Vida

Docentes: Dr. Dario Rodriguez
Mg. Hernán Amatriain
Lic. Santiago Bianco
Lic. Sebastian Martins

Índice

1. INTRODUCCIÓN A LOS MODELOS DE CICLOS DE VIDA

No existe un único Modelo de Ciclo de Vida que defina los estados por los que pasa cualquier producto software. Dado que existe una gran variedad de aplicaciones para las que se construyen productos software (software de tiempo real, de gestión, de ingeniería y científico, empotrado, de sistemas, de computadoras personales, etc.) y que dicha variedad supone situaciones totalmente distintas, es natural que existan diferentes Modelos de Ciclo de Vida. Por ejemplo, en aquellos casos en que el problema sea perfectamente conocido, el grupo de desarrollo tenga experiencia en sistemas del mismo tipo, el usuario sea capaz de describir claramente sus requisitos, un ciclo de vida tradicional, en cascada o secuencial sería el adecuado. Por el contrario, si el desarrollo conlleva riesgos (sean técnicos o de otro tipo), un ciclo de vida en espiral será el más apropiado. Sin embargo, si se está ante el caso en que es necesario probarle el producto al usuario para demostrar la utilidad del mismo, se estará ante un ciclo de vida con prototipado, etc.

Un ciclo de vida debe:

- Determinar el orden de las fases del Proceso Software
- Establecer los criterios de transición para pasar de una fase a la siguiente

A continuación, se presentan los diferentes modelos de ciclo de vida más representativos: cascada, espiral y prototipado. No existe un Modelo de Ciclo de Vida que sirva para cualquier proyecto, esto debe quedar claro, cada proyecto debe seleccionar un ciclo de vida que sea el más adecuado para su caso. El ciclo de vida apropiado se elige en base a:

- la cultura de la corporación,
- el deseo de asumir riesgos,
- el área de aplicación,
- la volatilidad de los requisitos,
- y hasta qué punto se entienden bien dichos requisitos.

El ciclo de vida elegido ayuda a relacionar las tareas que forman el proceso software de cada proyecto.

2. APROXIMACIÓN TRADICIONAL

2.1. MODELO DE CICLO DE VIDA EN CASCADA

Este modelo fue presentado por primera vez por Royce en 1970. Se representa, frecuentemente, como un simple modelo con forma de cascada de las etapas del software, como muestra la Figura 2.1. En este modelo la evolución del producto software procede a través de una secuencia ordenada de transiciones de una fase a la siguiente según un orden líneal. Tales modelos semejan una máquina de estados finitos para la descripción de la evolución del producto software. El modelo en cascada ha sido útil para ayudar a estructurar y gestionar grandes proyectos de desarrollo de software dentro de las organizaciones.

Este modelo permite iteraciones durante el desarrollo, ya sea dentro de un mismo estado, ya sea de un estado hacia otro anterior, como muestran las flechas ascendentes de la Figura 2.1. La mayor iteración se produce cuando una vez terminado el desarrollo y cuando se ha visto el software producido, se decide comenzar de nuevo y redefinir los requisitos del usuario.

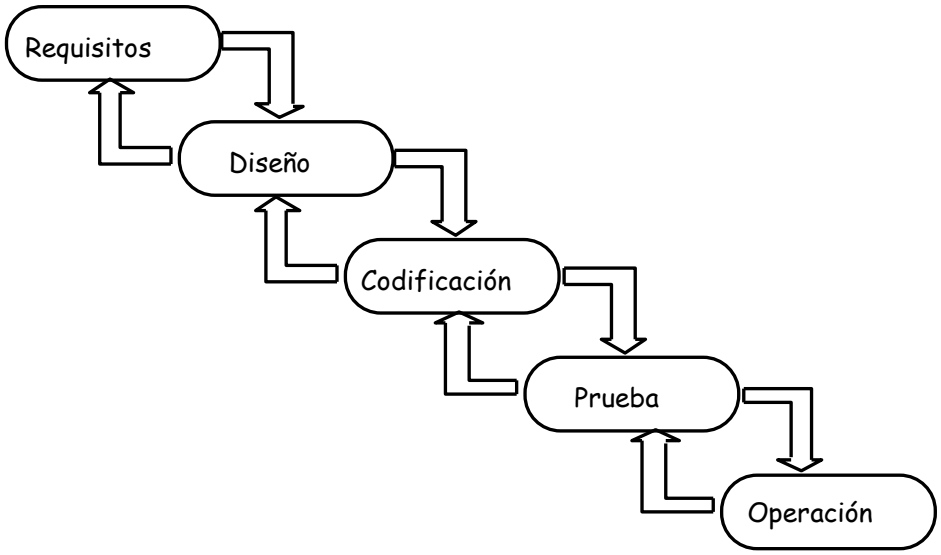

Figura 2.1. Ciclo de vida en cascada

El uso del modelo en cascada:

- Obliga a especificar lo que el sistema debe hacer (o sea, definir los requisitos) antes de construir el sistema (esto es, diseñarlo).
- Obliga a definir cómo van a interactuar los componentes (o sea, diseñar) antes de construir tales componentes (o sea codificar).

- Permite al líder del proyecto seguir y controlar los progresos de un modo más exacto. Esto le permite detectar y resolver las desviaciones sobre la planificación inicial.
- Requiere que el proceso de desarrollo genere una serie de documentos que posteriormente pueden utilizarse para la validación y el mantenimiento del sistema.

A menudo, durante el desarrollo, se pueden tomar decisiones que den lugar a diferentes alternativas, el modelo en cascada no reconoce esta situación. Por ejemplo, dependiendo del análisis de requisitos se puede implementar el sistema desde cero, o adoptar uno ya existente, o comprar un paquete que proporcione las funcionalidades requeridas. Es decir, resulta demasiado estricto para la flexibilidad que necesitan algunos desarrollos.

El modelo en cascada asume que los requisitos de un sistema pueden ser congelados antes de comenzar el diseño.

La figura 2.2. muestra la constante evolución de las necesidades del usuario. Se han representado en un espacio de tiempo/funcionalidad: según pasa el tiempo, aumentan las expectativas de funcionalidades que el usuario espera que tenga el sistema. La evolución está simplificada, pues ni mucho menos es lineal ni continua, pero para hacerse una idea es suficiente.

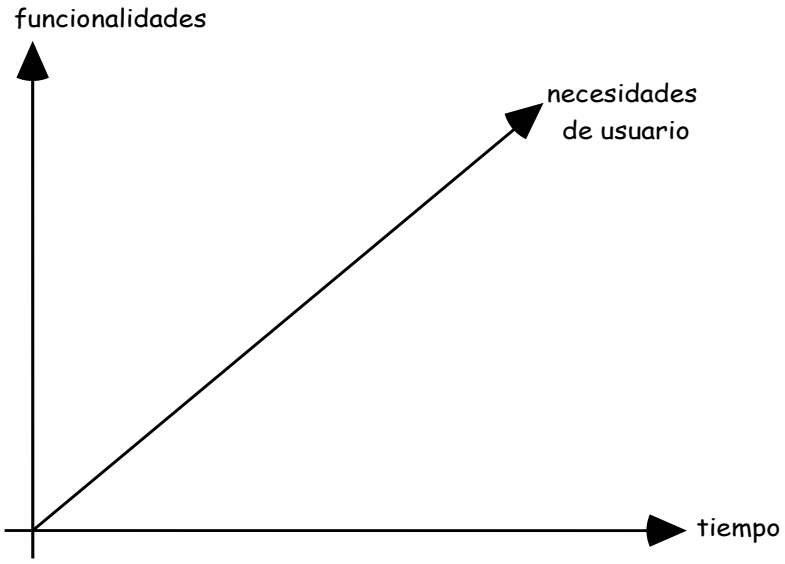

Figura 2.2. Evolución constante de las necesidades del usuario

El ciclo de vida en cascada tiene tres propiedades muy positivas:

- Las etapas están organizadas de un modo lógico. Es decir, si una etapa no puede llevarse a cabo hasta que se hayan tomado ciertas decisiones de más alto nivel, debe esperar hasta que esas decisiones estén tomadas. Así, el diseño espera a los requisitos, el código espera a que el diseño esté terminado, etc.
- Cada etapa incluye cierto proceso de revisión, y se necesita una aceptación del producto antes de que la salida de la etapa pueda usarse. Este ciclo de vida está organizado de modo que se pase el menor número de errores de una etapa a la siguiente.
- El ciclo es iterativo. A pesar de que el flujo básico es de arriba hacia abajo, el ciclo de vida en cascada reconoce, como ya se ha comentado, que los problemas encontrados en etapas inferiores afectan a las decisiones de las etapas superiores.

Existe una visión alternativa del modelo de ciclo de vida en cascada, mostrada en la figura 2.3, que enfatiza en la validación de los productos, y de algún modo en el proceso de composición existente en la construcción de sistemas software.

El proceso de análisis o descomposición subyacente en la línea superior del modelo de la figura 2.3 consiste en: los requisitos del sistema global se dividen en requisitos del hardware y requisitos del software. Estos últimos llevan al diseño preliminar de múltiples funciones, cada una de las cuales se expande en el diseño detallado, que, a su vez, evoluciona aún en un número mayor de programas unitarios. Sin embargo, el ensamblaje del producto final fluye justo en sentido contrario, dentro de un proceso de síntesis o composición. Primero se aceptan los programas unitarios probados. Entonces, éstos se agrupan en módulos que, a su vez, deben ser aceptados una vez probados. Los módulos se agrupan para certificar que el grupo formado por todos ellos incluyen todas las funcionalidades deseadas. Finalmente, el software es integrado con el hardware hasta formar un único sistema informático que satisface los requisitos globales.

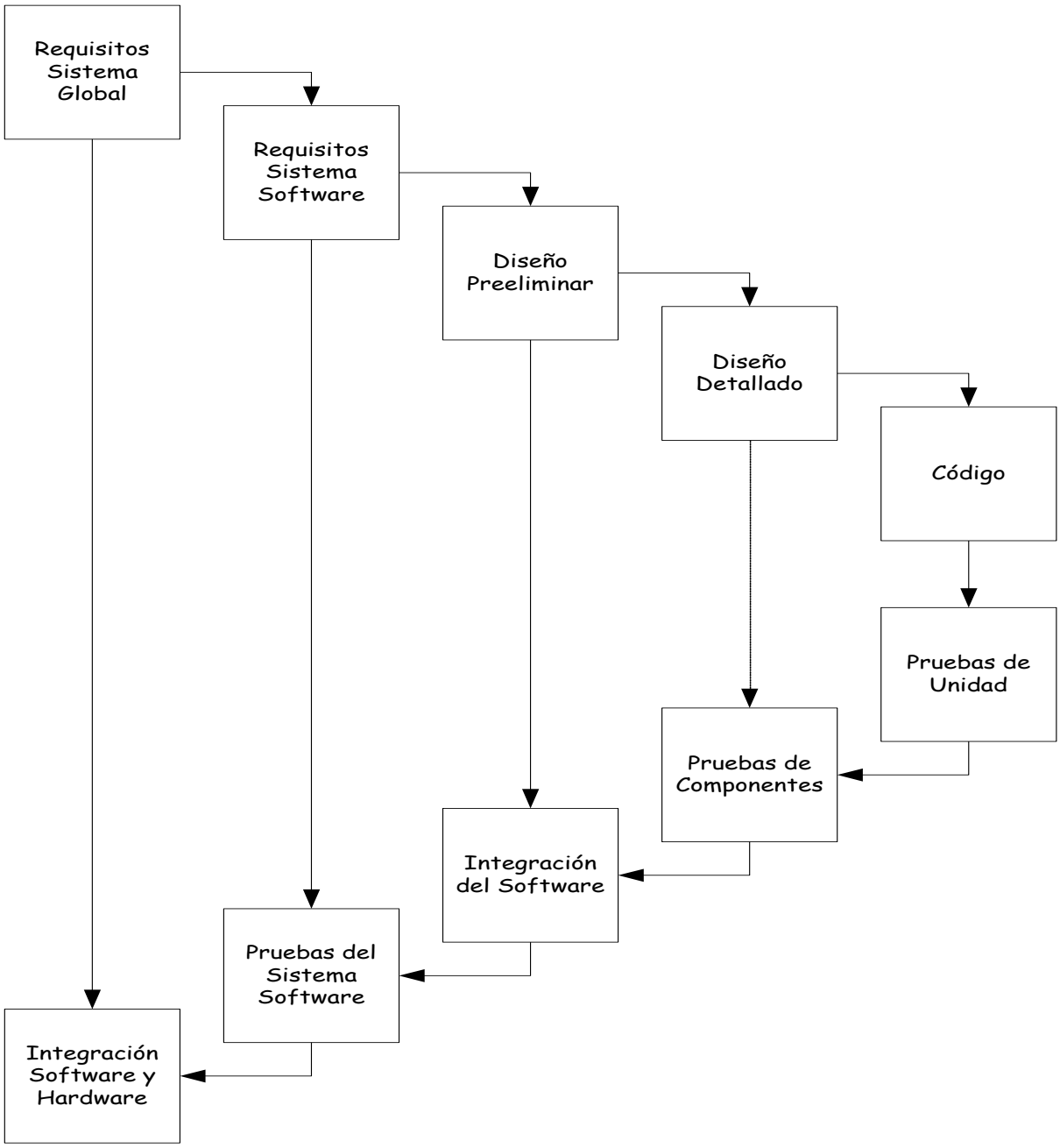

Figura 2.3. Visión alternativa del ciclo de vida en cascada

2.2. PROTOTIPADO

Suele ocurrir que, en los proyectos software, el cliente no tiene una idea muy detallada de lo que necesita, o que el ingeniero de software no está muy seguro de la viabilidad de la solución que tiene en mente. En estas condiciones, la mejor aproximación al problema es la realización de un prototipo.

El modelo de desarrollo basado en prototipos tiene como objetivo contrarestar el problema ya comentado del modelo en cascada: la congelación de requisitos mal comprendidos. La idea básica es que el prototipo ayude a comprender los requisitos del usuario. El prototipo debe incorporar un subconjunto de la función requerida al software, de manera que se puedan apreciar mejor las características y posibles problemas.

La construcción del prototipo sigue el ciclo de vida estándar, sólo que su tiempo de desarrollo será bastante más reducido, y no será muy rigurosa la aplicación de los estándares.

El problema del prototipo es la elección de las funciones que se desean incorporar, y cuáles son las que hay que dejar fuera, pues se corre el riesgo de incorporar características secundarias, y dejar de lado alguna característica importante. Una vez creado el prototipo, se le enseña al cliente, para que "juegue" con él durante un período de tiempo, y a partir de la experiencia aportar nuevas ideas, detectar fallos, etc.

Cuando se acaba la fase de análisis del prototipo, se refinan los requisitos del software, y a continuación se procede al comienzo del desarrollo a escala real. En realidad, el desarrollo principal se puede haber arrancado previamente, y avanzar en paralelo, esperando para un tirón definitivo a la revisión del prototipo.

El ciclo de vida clásico queda modificado de la siguiente manera por la introducción del uso de prototipos:

1.- Análisis preliminar y especificación de requisitos
2.- Diseño, desarrollo e implementación del prototipo
3.- Prueba del prototipo
4.- Refinamiento iterativo del prototipo
5.- Refinamiento de las especificaciones de requisitos
6.- Diseño e implementación del sistema final

Existen tres modelos derivados del uso de prototipos:

- *Maqueta.* Aporta al usuario ejemplo visual de entradas y salidas. La diferencia con el anterior es que en los prototipos desechables se utilizan datos reales, mientras que las maquetas son formatos encadenados de entrada y salida con datos simples estáticos.
- *Prototipo desechable.* Se usa para ayudar al cliente a identificar los requisitos de un nuevo sistema. En el prototipo se implantan sólo aquellos

aspectos del sistema que se entienden mal o son desconocidos. El usuario, mediante el uso del prototipo, descubrirá esos aspectos o requisitos no captados. Todos los elementos del prototipo serán posteriormente desechados.

- *Prototipo evolutivo.* Es un modelo de trabajo del sistema propuesto, fácilmente modificable y ampliable, que aporta a los usuarios una representación física de las partes claves del sistema antes de la implantación. Una vez definidos todos los requisitos, el prototipo evolucionará hacia el sistema final. En los prototipos evolutivos, se implantan aquellos requisitos y necesidades que son claramente entendidos, utilizando diseño y análisis en detalle así como datos reales.

2.3. MODELO EN ESPIRAL

El modelo en espiral para el desarrollo de software representa un enfoque dirigido por el riesgo para el análisis y estructuración del proceso software. Fue presentado por primera vez por Böehm en 1986. El enfoque incorpora métodos de proceso dirigidos por las especificaciones y por los prototipos. Esto se lleva a cabo representando ciclos de desarrollo iterativos en forma de espiral, denotando los ciclos internos del ciclo de vida análisis y prototipado precoz, y los externos, el modelo clásico. La dimensión radial indica los costes de desarrollo acumulativos y la angular el progreso hecho en cumplimentar cada desarrollo en espiral. El análisis de riesgos, que busca identificar situaciones que pueden causar el fracaso o sobrepasar el presupuesto o plazo, aparecen durante cada ciclo de la espiral. En cada ciclo, el análisis del riesgo representa groseramente la misma cantidad de desplazamiento angular, mientras que el volumen desplazado barrido denota crecimiento de los niveles de esfuerzo requeridos para el análisis del riesgo como se ve en la figura 2.4.

La primera ventaja del modelo en espiral es que su rango de opciones permiten utilizar los modelos de proceso de construcción de software tradicionales, mientras su orientación al riesgo evita muchas dificultades. De hecho, en situaciones apropiadas, el modelo en espiral proporciona una combinación de los modelos existentes para un proyecto dado. Otras ventajas son:

- Se presta atención a las opciones que permiten la reutilización de software existente.
- Se centra en la eliminación de errores y alternativas poco atractivas.
- No establece una diferenciación entre desarrollo de software y mantenimiento del sistema.
- Proporciona un marco estable para desarrollos integrados hardware-software.

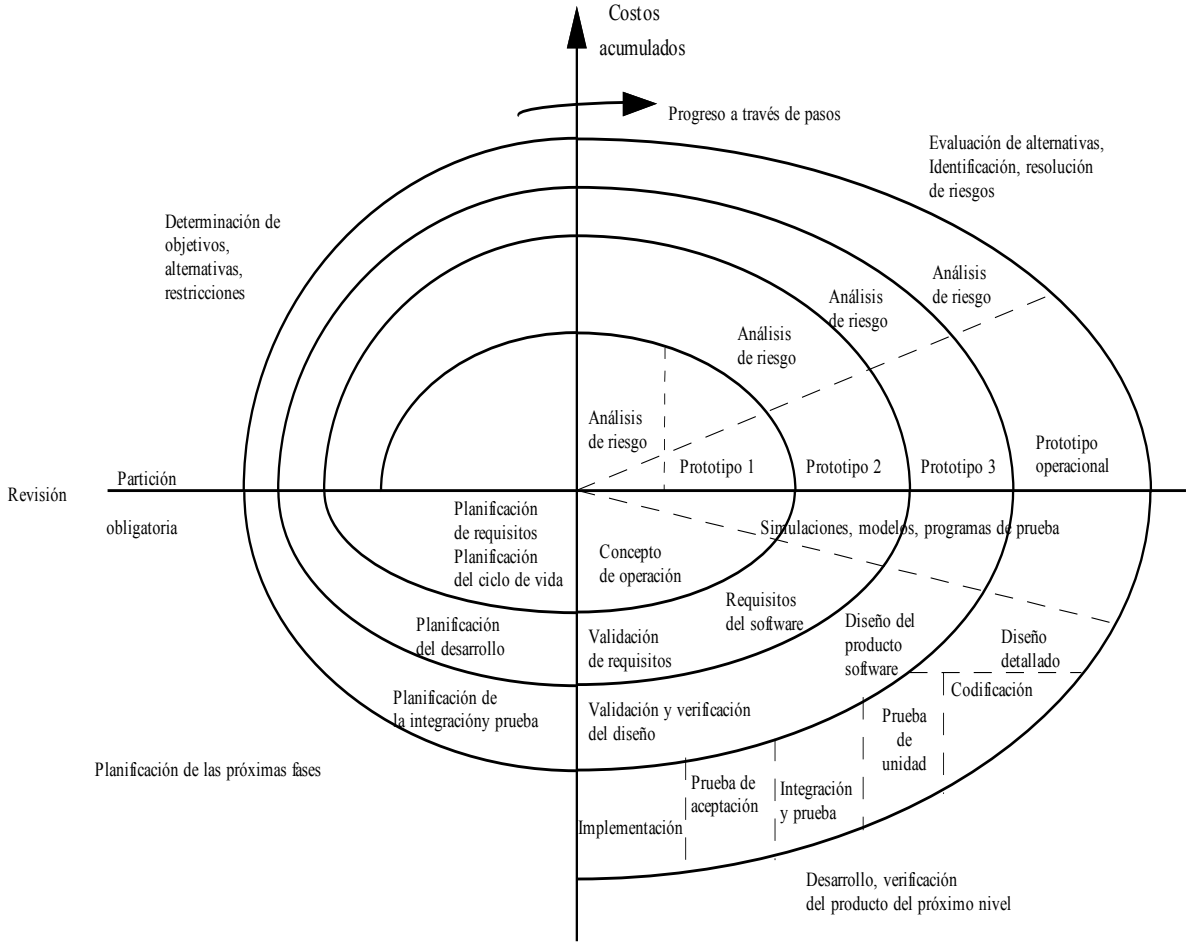

Figura 2.4. Modelo en Espiral

En realidad el modelo en espiral no significa una visión radicalmente distinta de los modelos tradicionales, o prototipado. Cualquiera de los modelos pueden verse con una representación espiral. Este modelo de Böehm es más bien una formalización o representación de los modelos de ciclo de vida más acertada que la representación en forma de cascada, pues permite observar mejor todos los elementos del proceso (incluido riesgos, objetivos, etc.).

3. VENTAJAS DE DEFINIR UN PROCESO SOFTWARE

Un proyecto sin estructura es un proyecto inmanejable. No puede ser planificado, ni estimado, ni su progreso ser controlado, y mucho menos alcanzar un compromiso de costes o tiempos. La idea originaria de buscar ciclos de vida que describan los estados por los que pasa el producto, o procesos software que describan las actividades a realizar para transformar el producto, surge de la necesidad de tener un esquema que sirva como base para planificar, organizar, asignar personal, coordinar, presupuestar, y dirigir las actividades de la construcción de software. De hecho, muchos proyectos han terminado mal porque las fases de desarrollo se realizaron en un orden erróneo.

Por tanto, al comienzo de un proyecto software, se debe elegir el Ciclo de Vida que seguirá el producto a construir, en base a las consideraciones del apartado anterior. El Modelo de Ciclo de Vida elegido llevará a *encadenar* las tareas y actividades del Proceso Software de una determinada manera: algunas tareas no será necesario realizarlas, otras deberán realizarse más de una vez, etc. Una vez conseguido un Proceso Software concreto para el proyecto en cuestión, se está preparado para planificar los plazos del proyecto, asignar personas a las distintas tareas, presupuestar los costos del proyecto, etc.

Concretamente, los procesos software se usan como:

- Guía prescriptiva sobre la documentación a producir para enviar al cliente.
- Base para determinar qué herramientas, técnicas y metodologías de Ingeniería de Software serán más apropiadas para soportar las diferentes actividades.
- Marco para analizar o estimar patrones de asignación y consumo de recursos a lo largo de todo el ciclo de vida del producto software.
- Base para llevar a cabo estudios empíricos para determinar qué afecta a la productividad, el coste y la calidad del software.
- Descripción comparativa sobre cómo los sistemas software llegan a ser lo que son.

4. ESTANDAR IEEE SOBRE PROCESO SOFTWARE

4.1. INTRODUCCIÓN

A continuación, se describen en detalle las fases o subprocesos que conforman el Proceso Base de Construcción del Software y que se corresponden con el estándar IEEE 1074 [IEEE 1074, 1989]. Cada subproceso se detalla a nivel de propósito, actividades involucradas y documentación principal propuesta por el estándar. El estándar IEEE 1074-1989 determina el conjunto de actividades esenciales, no ordenadas en el tiempo, que deben ser incorporadas dentro de un Modelo de Ciclo de Vida del producto software. Este modelo es seleccionado y establecido por el usuario para el proyecto a desarrollar, ya que la norma no define un ciclo de vida en particular.

El estándar, además, incluye la siguiente información que no se trata aquí:

- Descripción de cada actividad.
- Informe de entrada y salida para cada actividad.
- Fuente y destino de la información a nivel de proceso y de actividad, que refleja la relación entre los procesos.
- Productos obtenidos por el desarrollo de cada actividad.

El estándar ha sido escrito por organizaciones responsables de la gestión y desarrollo del software. Está dirigido a los gestores de proyectos, a los desarrolladores de software, a los responsables de la garantía de la calidad, a quienes ejecutan tareas de apoyo, a los usuarios y al personal de mantenimiento.

4.2. DESCRIPCIÓN GLOBAL DEL PROCESO

El Proceso Base de Construcción de Software consiste en analizar las necesidades de la organización en un dominio, desarrollar una solución que las satifaga y posteriormente reinsertar la solución en el dominio, bajo un marco de gestión, seguimiento, control y gestión de la calidad. Esta relación del Proceso Base con su entorno organizativo se representa en la figura 4.1.

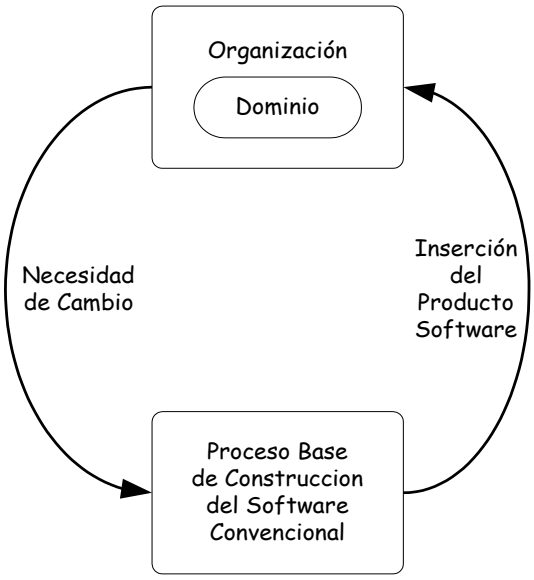

Figura 4.1. Relación del Proceso Base de Construcción de Software con su entorno organizativo

El proceso software está compuesto, a su vez, de cuatro procesos principales tal y como se muestra en la figura 4.2., cada uno de los cuales agrupa una serie de actividades que se encargan de la realización de sus requisitos asociados. Estos procesos son los siguientes:

- **Proceso de Selección de un Modelo de Ciclo de Vida del Producto:** identifica y selecciona un ciclo de vida para el software que se va a construir.
- **Procesos de Gestión del Proyecto:** crean la estructura del proyecto y aseguran el nivel apropiado de la gestión del mismo durante todo el ciclo de vida del software.
- **Procesos Orientados al Desarrollo del Software:** producen, instalan, operan y mantienen el software y lo retiran de su uso. Se clasifican en procesos de pre-desarrollo, desarrollo y post-desarrollo.
- **Procesos Integrales del Proyecto:** son necesarios para completar con éxito las actividades del proyecto software. Aseguran la terminación y calidad de las funciones del mismo. Son simultáneos a los procesos orientados al desarrollo del software e incluyen actividades de no desarrollo.

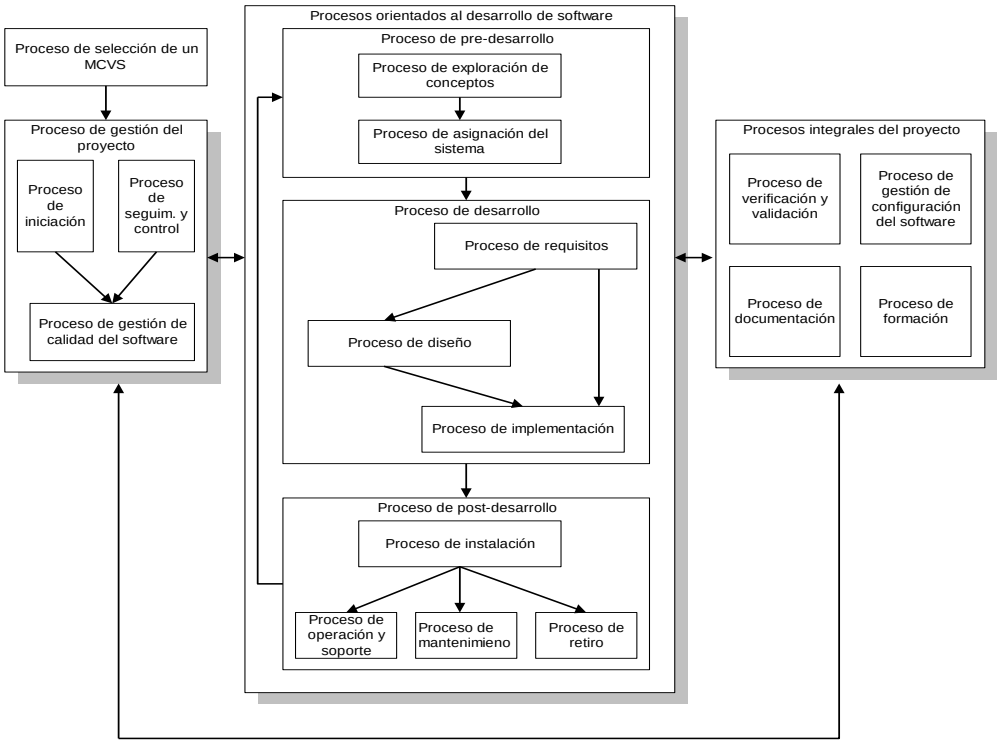

Figura 4.2. Modelo de proceso software (IEEE 1989)

4.3. PROCESO DE SELECCIÓN DE UN MODELO DE CICLO DE VIDA DEL SOFTWARE

Durante este proceso se selecciona un ciclo de vida que establece el orden de ejecución de las distintas actividades marcadas por el proceso e involucradas en el proyecto. Como ya se ha visto, en base al tipo de producto software a desarrollar y a los requisitos del proyecto se identifican y analizan posibles modelos de ciclo de vida para dicho proyecto y se selecciona un único modelo que lo soporte adecuadamente. El estándar no dictamina ni define un ciclo de vida del software específico, ni una metodología de desarrollo, sólo requiere elegir y seguir un modelo de ciclo de vida.

La información de salida de este proceso es el **Modelo de Ciclo de Vida del Software** seleccionado

Actividades a realizar:

- Identificar los posibles modelos de ciclo de vida del software.
- Seleccionar el modelo más adecuado para el proyecto.

Documentos de salida:

- Modelo de ciclo de vida seleccionado.

4.4. PROCESOS DE GESTIÓN DEL PROYECTO

La gestión del proyecto presupone establecer condiciones para el desarrollo del mismo. Involucra actividades de planificación, estimación de recursos, seguimiento y control, y evaluación del proyecto.

La **planificación** de proyectos se define como la predicción de la duración de las actividades y tareas a nivel individual, los recursos requeridos, la concurrencia y solapamiento de tareas para ser desarrollados en paralelo y el camino crítico a través de la red de actividades.

La **estimación** se define como la predicción de personal, esfuerzo y costos que se requerirá para terminar todas las actividades y productos conocidos asociados con el proyecto.

La determinación del tamaño del producto a desarrollar es una de las primeras tareas en la gestión del proyecto, ya que sin unos conocimientos razonables, es imposible planificar o estimar el esfuerzo involucrado. El tamaño se define como la cantidad de código fuente, especificaciones, casos de prueba, documentación del usuario y otros productos tangibles que son salida del proyecto. La determinación del tamaño se basa principalmente en la experiencia de proyectos anteriores.

El seguimiento de proyectos es la recolección de datos y su acumulación sobre recursos consumidos, costos generados, e hitos asociados con un proyecto. **Medir** en un proyecto se define como el registro de todos los productos generados en el mismo, de todos los recursos requeridos, planificación y solapamiento de todas las actividades y tareas y de todos los factores que impactan en el proyecto (conocimientos, métodos, herramientas, lenguajes, limitaciones, problemas y el entorno físico).

La medición en los proyectos de desarrollo de software es una actividad fundamental para la mejora de la productividad, el costo y la calidad del producto final. La medición, entrada a estudios empíricos, es el único modo para detectar fallos en el proceso de construcción de software.

PROCESO DE INICIACIÓN DEL PROYECTO

Abarca aquellas actividades de creación de la estructura del proyecto. Durante este proceso se define el ciclo de vida del software (asignación de responsables de cada actividad) para este proyecto y se establecen los planes para su gestión. Se estiman y asignan los recursos; esto consiste en determinar los costos y recursos necesarios a fin de ejecutar las distintas tareas que demanda el proyecto. Se identifican y seleccionan estándares, metodologías y herramientas para la gestión y ejecución del mismo y, por último, se prepara y establece un plan para su implementación adecuada y oportuna, incluyendo hitos y revisiones.

El **Plan de Gestión del Proyecto Software** que conducirá el desarrollo se produce como culminación de este proceso.

Actividades a realizar:

- Establecer el mapa de actividades para el modelo de ciclo de vida del software seleccionado.
- Asignar los recursos del proyecto.
- Definir el entorno del proyecto.
- Planificar la gestión del proyecto.

Documentos de salida:

- Plan de Gestión del Proyecto.
- Plan de Retiro.

Técnicas a utilizar:

- Análisis de Camino Crítico (CPM).
- Análisis PERT.
- Diagrama de GANTT.
- Técnicas Estadísticas.
- Técnicas de Simulación (Método de MONTECARLO).
- Puntos de Función.
- Modelos Empíricos de Estimación (COCOMO, PUTMAN).
- Técnicas de Descomposición para Estimación.

PROCESO DE SEGUIMIENTO Y CONTROL DEL PROYECTO

Es un proceso iterativo de seguimiento, registro y gestión de costos, problemas, y rendimiento de un proyecto durante su ciclo de vida. En este proceso se realiza un análisis de riesgos (técnico, económico, operativo y de soporte, y del programa o calendario) que permite identificar los problemas potenciales, determinar su

probabilidad de ocurrencia y su impacto y establecer los pasos para su gestión. Los riesgos identificados y su gestión se documentan en el **Plan de Contingencia.**

El progreso de un proyecto se revisa y mide con respecto a los hitos establecidos en el Plan de Gestión del Proyecto Software.

Actividades a realizar:

- Analizar riesgos.
- Realizar la planificación de contingencias.
- Gestionar el proyecto.
- Archivar registros.
- Implementar el Sistema de Informes de Problemas.

Documentos de salida:

- Análisis de riesgos.
- Plan de contingencias.
- Registro histórico de proyectos.

Técnicas a utilizar:

- Análisis de Riesgo Técnico.
    - Modelización y Simulación Estática y Dinámica.
    - Prototipado.
    - Revisiones.
    - Auditorías.
- Análisis de Riesgo Económico.
- Análisis de Finanzas.
- Retorno de la Inversión.
- Análisis de Riesgo Operativo y de Soporte.
- Análisis de Riesgo del Programa.
    - Análisis de Camino Crítico (CPM).
    - Técnicas de Nivelación de Recursos.

PROCESO DE GESTIÓN DE LA CALIDAD DEL SOFTWARE

Su objetivo es la planificación y administración de las acciones necesarias para proveer una confianza adecuada en la calidad de los productos software; es decir, que satisfagan los requisitos técnicos establecidos.

El proceso de Gestión de la Calidad del Software se documenta en un **Plan de Garantía de la Calidad del Software**. Sus actividades abarcan el ciclo de vida completo del software. Para abordar este proceso de protección del software, con una visión global, se consideran los siguientes tres aspectos principales:

- Métricas del Software para el control del proyecto.
- Verificación y Validación, incluyendo pruebas y procesos de revisión.
- Gestión de la Configuración del producto software.

Las **Métricas del Software** se definen como la aplicación continua de técnicas basadas en las medidas de los procesos de desarrollo del software y de sus productos para producir una información de gestión significativa y a tiempo, a la vez que se mejoran los procesos y sus productos.

La Verificación y Validación del Software involucra actividades imprescindibles para el control de la calidad del software. Se entiende por **Verificación** al conjunto de actividades para la comprobación de que un producto software está técnicamente bien construido; es decir, que el producto funciona (¿está el producto correctamente construido?). En general, comprobar si los productos construidos en una fase del ciclo de vida satisfacen los requisitos establecidos en la fase anterior, decidiendo si el producto, hasta el momento, es consistente y completo. De modo complementario la **Validación** trata la comprobación de que el producto software construido es el que se deseaba construir; es decir, que el producto funciona como el usuario quiere y hace las funciones que se establecieron (¿el producto construido es el correcto?). En general, comprobar si el software construido satisface los requisitos del usuario. Evidentemente, sólo tiene objeto validar el producto que está verificado (no interesa comprobar si un producto que no funciona es lo que se pedía).

Tanto el proceso de Verificación y Validación, como el proceso de Gestión de la Configuración del software se tratan posteriormente, dentro de los procesos integrales.

Actividades a realizar:

- Planificar la garantía de la calidad del software.
- Desarrollar métricas de calidad.
- Gestionar la calidad del software.
- Identificar necesidades de mejora de la calidad.

Documentos de salida:

- Plan de garantía de calidad del software.
- Recomendaciones de mejora de calidad software.

Técnicas a aplicar:

- Técnicas de Planificación y Estimación.
- Métricas de Calidad del Software.

4.5. PROCESOS DE PRE-DESARROLLO

Los procesos orientados al desarrollo de software, sean pre, en o pos, se estudian en el Módulo del curso llamada "Técnicas de Ingeniería de Software"
Son los procesos que se deben realizar antes de que comience el desarrollo propiamente dicho del software. El esfuerzo de desarrollo se inicia con la identificación de una necesidad de automatización. Esta necesidad, para ser satisfecha, puede requerir una nueva aplicación, o un cambio de todo o parte de una aplicación existente. El pre-desarrollo abarca desde el reconocimiento del problema hasta la determinación de los requisitos funcionales a nivel de sistema, pasando por el estudio de la viabilidad de su solución automatizada.

PROCESO DE EXPLORACIÓN DE CONCEPTOS

Este proceso incluye la identificación de una necesidad, la formulación de soluciones potenciales, su evaluación (estudio de viabilidad) y refinamiento a nivel de sistema. Una vez establecidos sus límites, se genera el **Informe de la Necesidad** del sistema a desarrollar. Este informe inicia el Proceso de Asignación del Sistema y, o, el Proceso de Requisitos, y alimenta los Procesos de Gestión del Proyecto.

El Informe de la Necesidad es un documento que constituye la base de todo el trabajo de ingeniería posterior.

Actividades a realizar:

- Identificar ideas o necesidades.
- Formular soluciones potenciales.
- Conducir estudios de viabilidad.
- Planificar la transición del sistema (si se aplica).
- Refinar y Finalizar la idea o necesidad.

Documentos de salida:

- Modelo de la situación actual.
- Modelo del dominio del problema.
- Informe preliminar de necesidades.
- Soluciones alternativas posibles.
- Soluciones recomendadas.
- Plan de transición.
    - Informe del impacto de la transición.

Técnicas a usar:

- Técnicas de Adquisición de Conocimientos.
- Análisis Económico (Coste/Beneficio).
- Análisis Técnico.
- Análisis Alternativos.
- Técnicas de Modelización.
- Diagramas de Flujos de Datos (DFD).
- Prototipado.

PROCESO DE ASIGNACIÓN DEL SISTEMA

Este proceso se realiza cuando el sistema requiere tanto del desarrollo de hardware como de software, o cuando no se puede asegurar que sólo se necesita desarrollo de software. El Informe de la Necesidad se analiza para identificar las entradas, el procesamiento que se aplica a la entrada, las salidas requeridas y las funciones del sistema total, que permiten desarrollar la arquitectura del sistema e identificar las funciones del hardware, del software y de los interfaces. Este proceso culmina con la **Especificación de Requisitos del Software**, la **Especificación de Requisitos del Hardware** y la **Especificación del Interfaz del Sistema**.

Debido a que el software es parte de un sistema mayor, se comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software, para su análisis y refinamiento en el Proceso de Requisitos. Este planteamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos, tales como hardware, personas y bases de datos. Antes de la asignación del sistema, el análisis abarca los requisitos globales a nivel de sistema con una pequeña cantidad de análisis y de diseño a un nivel superior.

El análisis de sistemas requiere una comunicación intensa entre el cliente y el analista. El cliente debe comprender los objetivos del sistema y ser capaz de exponerlos claramente. El analista debe saber qué preguntas hacer, qué consejos dar y qué investigación realizar. Si la comunicación se rompe, el éxito del proyecto entero estará en peligro.

En el análisis del sistema (Proceso de Exploración de Conceptos) se definen los objetivos del mismo, la información que se va a obtener, la información que se va a suministrar, las funciones, el comportamiento y el rendimiento requerido. El analista se asegura de distinguir entre lo que "necesita" el cliente (elementos críticos para la realización) y lo que el cliente "quiere" (elementos deseables pero no esenciales).

Una vez que la función, el rendimiento y los interfaces (determinados por el

comportamiento) están delimitados, se procede a realizar la tarea denominada asignación. Durante la asignación, las funciones son asignadas a uno o más elementos genéricos del sistema; es decir, software, hardware, personal, bases de datos, documentación, procedimientos. A menudo, se proponen y evalúan varias asignaciones. Esencialmente, lo que se hace es asignar a cada elemento del sistema un ámbito de funcionamiento y de rendimiento.

Asignadas las funciones del sistema informático, se puede crear un modelo que represente las interrelaciones, entre los distintos elementos del sistema y establezca una base para los posteriores pasos de análisis de requisitos y de diseño. Se representa el sistema definido mediante modelos de la arquitectura del sistema (salida, entrada, proceso y control, interfaz de usuario, y mantenimiento y autocomprobación). En primer lugar, se realiza un **Diagrama de Contexto** de la arquitectura (DC), que establece los límites de información entre los que se está implementando el sistema y el entorno en el que va a funcionar. En esencia, el DC ubica el sistema en el contexto de su entorno externo.

Se refina el diagrama de contexto de la arquitectura considerando con más detalle la función del sistema. Se identifican los subsistemas principales que permiten el funcionamiento del sistema considerado en el contexto definido por el DC. Se definen los subsistemas principales en un **Diagrama de Flujo** de la arquitectura (DF). El diagrama de flujo de la arquitectura muestra los subsistemas principales y las líneas importantes de flujo de información (control y datos). En esta etapa, cada uno de los subsistemas pueden contener uno o más elementos del sistema (por ejemplo: hardware, software, personal) según hayan sido asignados.

El diagrama inicial de flujo de la arquitectura se constituye en el nodo raíz de la jerarquía de DF. Se puede ampliar cada subsistema del DF inicial en otro diagrama de arquitectura dedicado exclusivamente a él. Este proceso de descomposición de arriba a abajo (top-down) permite disponer de una jerarquía de DF, donde cada uno de los DF del sistema se puede utilizar como punto de partida para los posteriores pasos de ingeniería del subsistema que describe.

Actividades a realizar:

- Analizar las funciones del sistema.
- Desarrollar la arquitectura del sistema.
- Descomponer los requisitos del sistema.

Documentos de salida:

- Especificación de requisitos funcionales del software.
- Especificación de requisitos funcionales del hardware.

- Especificación de la interfaz del sistema.
- Descripción funcional del sistema.
- Arquitectura del sistema.

Técnicas a utilizar:

- Técnicas de Adquisición de Conocimientos.
- Técnicas de Modelización.
- Diagramas de Flujo de Datos (DFD).

4.6. PROCESOS DE DESARROLLO

Son los procesos que se deben realizar para la construcción del producto software. Estos definirán qué información obtener y cómo estructurar los datos, qué algoritmos usar para procesar los datos y cómo implementarlos y qué interfaces desarrollar para operar con el software y cómo hacerlo.

A partir del Informe de la Necesidad, con el soporte de las actividades de los Procesos Integrales y bajo el Plan de Gestión del Proyecto, los Procesos de Desarrollo producen el software (código y documentación).

PROCESO DE REQUISITOS

Incluye las actividades iterativas dirigidas al desarrollo de la Especificación de Requisitos del Software. Para la determinación completa y consistente de los requisitos del software el análisis se enfatiza sobre la salida resultante, la descomposición de los datos, el procesamiento de los datos, las bases de datos (si existen) y los interfaces del usuario, del software y del hardware.

La **Especificación de Requisitos del Software** es el establecimiento conciso y preciso de un conjunto de requisitos que deben ser satisfechos por un producto software, indicando, siempre que sea apropiado, el procedimiento mediante el cual se puede determinar si se satisfacen los requisitos dados. Describe los requisitos funcionales, de rendimiento, y de interfaz del software y define los entornos de operación y de soporte. Este documento es la salida con que culmina este proceso.

Un requisito es una condición o característica que debe tener o cumplir un sistema o componente de un sistema para satisfacer un contrato, norma, especificación, u otro documento formalmente impuesto. El conjunto de todos los requisitos forma la base para el desarrollo consiguiente del sistema o componente del sistema.

Existen tres tipos de requisitos: funcionales, de rendimiento y de interfaz. Un **requisito funcional** especifica la función que un sistema o componente de un sistema debe ser capaz de realizar (por ejemplo: emitir facturas). Un **requisito de rendimiento** especifica una característica numérica tanto estática (por ejemplo: número de terminales del sistema, número de usuarios simultáneos que soporta el sistema, número de archivos y registros del mismo, etc.) como dinámica (por ejemplo: el 95% de las transacciones se deben procesar en menos de un segundo) que debe tener un sistema o componente de un sistema. Los **requisitos de interfaz** determinan las características que el software debe soportar para cada interfaz humano del producto software (interfaz de usuario), las características lógicas de cada interfaz entre el producto software y las componentes hardware del sistema (interfaz de hardware), el uso de otros productos software (por ejemplo: un sistema de gestión de base de datos, un sistema operativo, o un paquete matemático) e interfaces con otros sistemas de aplicación (interfaz de software). Por ejemplo: que la interfaz del usuario cuente con ventanas superpuestas.

Existen además algunos atributos del software (seguridad, consistencia, facilidad de traza, etc.) que pueden dar lugar a requisitos específicos del mismo, por ejemplo: que el acceso a ciertos datos sea restringido.

Actividades a realizar:

- Definir y desarrollar los requisitos del software.
- Definir los requisitos de interfaz.
- Priorizar e integrar los requisitos del software.

Documentos de salida:

- Especificación de requisitos del software.
- Requisitos del interfaz con el usuario.
- Requisitos del interfaz con otro software.
- Requisitos del interfaz con el hardware.
- Requisitos del interfaz con el sistema físico.

Técnicas a utilizar:

- Técnicas Orientadas a los Procesos:
    - Análisis Estructurado.
        - Diagramas de Flujos de Datos (DFD).
        - Diccionario de Datos (DD).
        - Especificación de Procesos Primitivos (EPP).
    - SADT (Structured Analyses and Design Tecniques).
        - Diagramas de Transición de Estados.

        - Diagramas de Descomposición.
            - WRS (Working Breakdown Structure).
            - RBS (Resources Breakdown Structure).
            - OBS (Object Breakdown Structure).
        - ACTIGRAMAS (Diagrama de Actividades).
    - Técnicas Orientadas a los Datos:
        - Diagramas de Entidad-Relación.
        - DATAGRAMAS (Diagramas de Datos).
    - Técnicas Orientadas a los Objetos:
        - Diagrama de Clases/Objetos.
        - Jerarquía de Clases/Objetos.
    - Técnicas Formales de Especificación
        - Técnicas Relacionales:
            - Ecuaciones Implícitas.
            - Relaciones Recurrentes.
            - Axiomas Algebraicos.
            - Expresiones Regulares.
        - Técnicas Orientadas al Estado.
            - Tablas de Decisión.
            - Tablas de Eventos.
            - Tablas de Transición.
            - Mecanismos de Estados Finitos.
            - Redes de Petri.
    - Técnicas de Prototipación.

PROCESO DE DISEÑO

Es el proceso central que unifica los procesos de desarrollo y de mantenimiento del software. Su objetivo es desarrollar una representación coherente y organizada del sistema software que satisfaga la Especificación de Requisitos del Software. La calidad de dicha representación se puede evaluar. El Proceso de Diseño traduce el "qué hacer" de las especificaciones de los requisitos en el "cómo hacerlo" de las especificaciones de diseño. Inicialmente, la representación describe una visión sistémica y holística del software. Posteriores, refinamientos de diseño conducen a una representación que se acerca al código fuente.

El diseño del software puede verse desde dos perspectivas: la técnica y la de gestión del proyecto. Desde el punto de vista técnico, el diseño comprende cuatro actividades: diseño de los datos, diseño arquitectónico, diseño procedimental y diseño de las interfaces. Desde el punto de vista de gestión del proyecto, el diseño va del diseño arquitectónico (diseño preliminar o diseño de alto nivel) al diseño

detallado (diseño de bajo nivel).

Desde el punto de vista de la gestión, el nivel de diseño arquitectónico o preliminar se focaliza en las funciones y estructuras de las componentes que conforman el sistema software. El nivel de diseño detallado se ocupa del refinamiento de la representación arquitectónica que lleva a una estructura de datos detallada y a las representaciones algorítmicas que se usan para cada componente modular del software.

El proceso de diseño del software comienza con la actividad de Realizar el Diseño Arquitectónico. Esta actividad genera la **Descripción de Diseño Arquitectónico del Software** en donde se describe el diseño de cada una de las componentes software, se especifican los datos, las relaciones y restricciones y se definen todos los interfaces externos (usuario, software y hardware) y los interfaces internos (entre las componentes).

La última actividad del proceso de diseño es Realizar el Diseño Detallado, donde se genera la **Descripción de Diseño del Software** (DDS) que especifica la estructura de los datos, los algoritmos y la información de control de cada componente software, y los detalles de los interfaces (usuario, hardware y software).

El diseño detallado se deriva del diseño preliminar, en consecuencia sus correspondientes actividades se realizan en secuencia mientras que el resto de las actividades de este proceso (analizar el flujo de información, diseñar la base de datos, diseñar los interfaces, seleccionar o desarrollar algoritmos) se ejecutan en paralelo. Estas últimas producen refinamientos del diseño que son también entradas a la actividad de diseño detallado.

Una actividad relevante es la de diseño de la base de datos. Esta comprende el diseño conceptual, lógico y físico de la base de datos. Los requisitos se modelizan dentro de un esquema externo que describe las entidades de datos, atributos, relaciones y restricciones. Los distintos esquemas externos se integran en un esquema conceptual único. El esquema conceptual se *aplica* entonces en un esquema lógico dependiente de la implementación. Finalmente, se definen las estructuras físicas de datos y los caminos de acceso. El resultado de esta actividad es la Descripción de la Base de Datos.

Como ya se ha mencionado, desde el punto de vista técnico y en el contexto de los diseños preliminar y detallado, se llevan a cabo varias actividades de diseño diferentes: diseño de datos, diseño arquitectónico, diseño procedimental y diseño del interfaz.

El diseño de datos transforma el modelo del campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software. El diseño arquitectónico define las relaciones entre los principales elementos estructurales del programa. El objetivo principal de este diseño es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos. Además, el diseño arquitectónico mezcla la estructura de programas y la estructura de datos y define los interfaces que facilitan el flujo de los datos a lo largo del programa. El diseño procedimental transforma los elementos estructurales en una descripción procedimental del software; se realiza después de que se ha establecido la estructura del programa y de los datos. El diseño del interfaz establece principalmente la disposición y los mecanismos para la interacción hombre-máquina.

La figura 4.3. representa las actividades del proceso de diseño y los documentos que las mismas producen. El documento de Descripción del Diseño del Software contiene los datos consolidados de la Descripción de la Arquitectura del Software, del Flujo de Información, de la Base de Datos, de las Interfaces y de los Algoritmos y forma parte de la configuración del software.

El diseño es el proceso en el que se asienta la calidad del desarrollo del software. El diseño produce las representaciones del software de las que puede evaluarse su calidad. El diseño es la única forma mediante la cual se puede traducir con precisión los requisitos del cliente en un producto o sistema acabado. El diseño de software sirve como base de todas las posteriores etapas del desarrollo y del proceso de mantenimiento. Sin diseño, se puede construir un sistema inestable; un sistema que falle cuando se realicen pequeños cambios; un sistema que pueda ser difícil de probar; un sistema cuya calidad no pueda ser evaluada hasta más adelante en el proceso de ingeniería del software, cuando quede poco tiempo y se haya gastado ya mucho dinero.

El proceso de diseño en la Ingeniería del Software conduce a un buen diseño mediante la aplicación de principios fundamentales de diseño (abstracción, refinamiento sucesivo, modularidad, estructura jerárquica de los módulos, estructura de los datos, jerarquía de control, procedimiento realizado por capas funcionales, ocultamiento de información), de métodos sistemáticos y de una adecuada revisión.

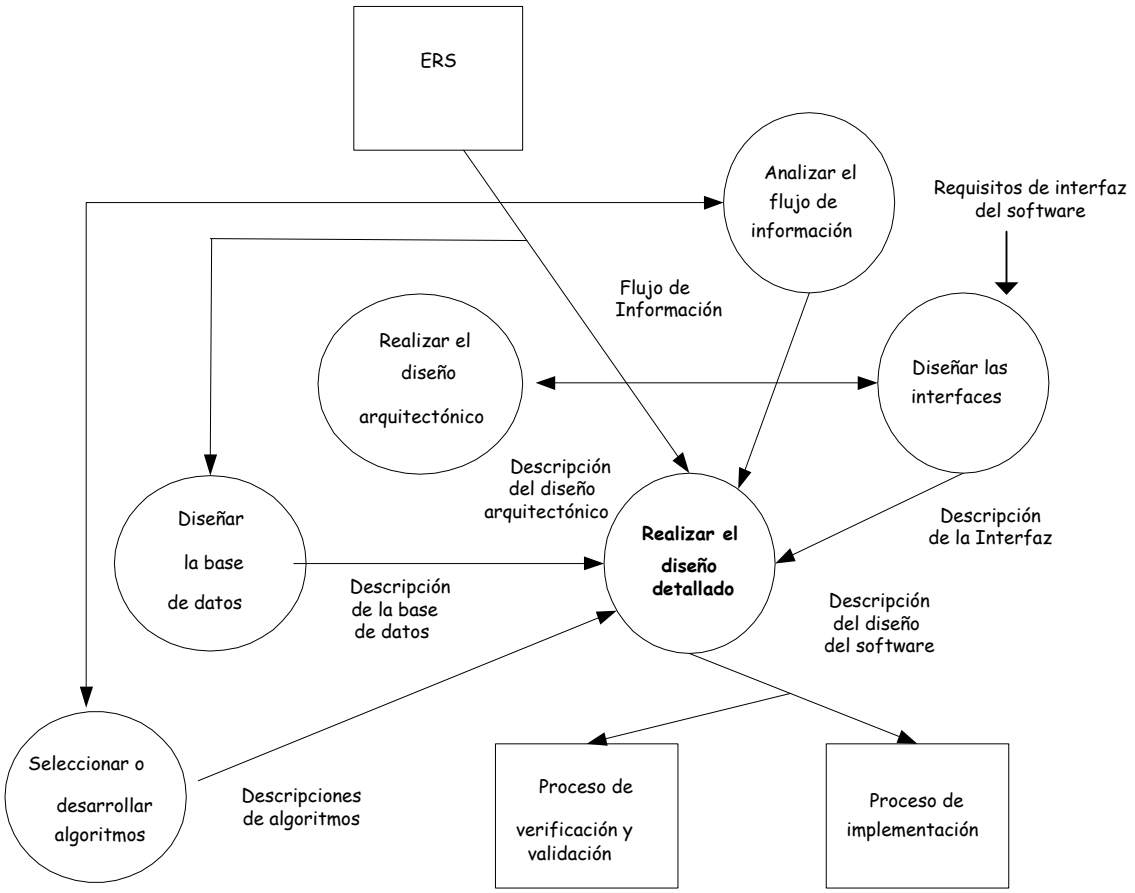

Figura 4.3. Representación del Proceso de Diseño

Para evaluar la calidad de una representación del diseño se deben tener en cuenta, entre otros, los siguientes criterios de calidad del diseño:

1. Un diseño debe exhibir una organización jerárquica que haga un uso inteligente del control entre los componentes del software.
2. Un diseño debe ser modular; esto es, el software debe estar dividido de forma lógica en elementos que realicen funciones y subfunciones específicas.
3. Un diseño debe contener representaciones distintas y separadas de los datos y de los procedimientos.
4. Un diseño debe llevar a módulos (por ejemplo: subrutinas o procedimientos) que exhiban características funcionales independientes.
5. Un diseño debe llevar a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.
6. Un diseño debe obtenerse mediante un método que sea reproducible y que esté conducido por la información obtenida durante el análisis de los requisitos del software

La modularidad se ha convertido en un enfoque ampliamente aceptado, ya que un diseño modular reduce la complejidad, facilita los cambios (un aspecto crítico de la facilidad de mantenimiento del software) y produce como resultado una implementación más sencilla permitiendo el desarrollo paralelo de las diferentes partes de un sistema.

En la arquitectura del software, para la definición de módulos se utiliza los conceptos de abstracción y de ocultamiento de información que derivan en el concepto de independencia funcional.

La independencia funcional se adquiere desarrollando módulos con una función definida y específica (máxima cohesión) y una aversión a una excesiva interacción con otros módulos (mínimo acoplamiento). Es decir, se trata de diseñar software de forma que cada módulo se centre en una subfunción específica de los requisitos y tenga un interfaz sencillo, cuando se ve desde otras partes de la estructura del software. Así, la independencia se mide con dos criterios cualitativos: la cohesión y el acoplamiento. La cohesión es una medida de la fortaleza funcional relativa de un módulo. El acoplamiento es una medida de la interdependencia relativa entre los módulos de una estructura de programa.

La notación de diseño, junto con los conceptos de la programación estructurada, permite al diseñador representar los detalles procedimentales, facilitando su traducción al código que puede realizarse automáticamente a través de una herramienta CASE (Computer Aided Software Engineering).

Actividades a realizar:

- Realizar el diseño arquitectónico.
- Analizar el flujo de información.
- Diseñar la base de datos (si se aplica).
- Diseñar los interfaces.
- Seleccionar o Desarrollar algoritmos (si se aplica).
- Realizar el diseño detallado.

Documentos de salida:

- Descripción de diseño del software.
- Descripción de la arquitectura del software.
- Descripción del flujo de información.
- Descripción de la base de datos.
- Descripción de las interfaces.
- Descripción de los algoritmos.

Técnicas a utilizar:

- Técnicas Orientadas a los Procesos
  - Diseño Estructurado.
    - Análisis de Transformación.
    - Análisis de Transacción.
  - Diseño del Diálogo de los Interfaces
    - Diseño Lógico o Diseño del Perfil.
    - HIPO (Hierarchy Input Process Output).
- Técnicas Orientadas a los Datos
  - Modelo Lógico de Datos.
  - Modelo Físico de Datos.
  - Warnier.
  - Jackson.
- Técnicas Orientadas a los Objetos
  - Modelo de Clases/Objetos.
  - Diagrama de Módulos.
- Técnicas de Diseño de Bajo Nivel
  - Programación Estructurada.
    - Diagramas Arborescentes.
    - Diagramas de Chapin.
  - Programación Orientada a Objetos.
    - Diagrama de Procesos.
    - Warnier.
    - Jackson (JSD - Jackson System Development).
- Técnicas de Prototipación.
- Técnicas de Refinamiento.

PROCESO DE IMPLEMENTACIÓN

Este proceso transforma la representación del diseño detallado de un producto software a una realización en un lenguaje de programación apropiado. El Proceso de Implementación produce el código fuente, el código de la base de datos (si se aplica) y la documentación, que constituyen la manifestación física del diseño de acuerdo a los estándares y metodologías del proyecto. Además, en este proceso se debe integrar el código y la base de datos. En el caso de que el sistema conste de componentes hardware y software, se debe planificar y realizar la integración del sistema. La salida de este proceso está sujeta a las pruebas de verificación y validación adecuadas. El código y la base de datos junto con la documentación producida durante los procesos previos son la primera representación completa del producto software.

Actividades a realizar:

- Crear los datos de prueba.
- Crear el código fuente.
- Generar el código objeto.
- Crear la documentación de operación.
- Planificar la integración.
- Realizar la integración.

Documentos de salida:

- Datos para las pruebas.
- Documentación del sistema.
- Documentación de usuario.
- Plan de integración.
- Sistema software integrado.

Técnicas a utilizar:

- Warnier.
- Jackson.
- Lenguajes de programación.

4.7. PROCESOS DE POST-DESARROLLO

Son los procesos que se deben realizar para instalar, operar, soportar, mantener y retirar un producto software. Se realizan después de la construcción del software. Es decir, se aplican a las últimas fases del ciclo de vida del software.

Una vez terminada la prueba del software, éste está casi preparado para ser entregado a los usuarios finales. Sin embargo, antes de la entrega se llevan a cabo una serie de actividades de garantía de calidad para asegurar que se han generado y catalogado los registros y documentos internos adecuados, que se ha desarrollado una documentación de alta calidad para el usuario y que se han establecido los mecanismos apropiados de control de configuraciones. Entonces, el software ya puede ser distribuido a los usuarios finales.

Tan pronto como se entregue el software a los usuarios finales, el trabajo del ingeniero del software cambia. En ese momento, el enfoque pasa de la construcción al mantenimiento; corrección de errores, adaptación al entorno y mejora de la funcionalidad. En todos los casos, la modificación del software no sólo afecta al código, sino también a la configuración entera; es decir, todos los documentos, datos y programas desarrollados en la fase de planificación y desarrollo.

PROCESO DE INSTALACIÓN

Implica el transporte y la instalación de un sistema software desde el entorno de desarrollo al entorno de destino. Incluye la carga, si es necesaria, de la base de datos, las modificaciones necesarias del software, las comprobaciones en el entorno de destino y la aceptación del cliente. Si durante la instalación surge un problema se identifica e informa acerca de él.

El Proceso de Instalación verifica que se implemente la configuración adecuada del software y culmina con la aceptación formal del mismo por parte del cliente conforme a lo especificado en el Plan de Gestión del Proceso Software y la realización con éxito de la prueba de aceptación del usuario.

Actividades a realizar:

- Planificar la instalación.
- Distribuir el software.
- Instalar el software.
- Cargar la base de datos (si se aplica).
- Aceptar el software en el entorno de operación.
- Realizar las actualizaciones (instalar el software probado).

Documentos de salida:

- Plan de instalación del software.
- Informe de instalación.

PROCESO DE OPERACIÓN Y SOPORTE

Involucra la operación del sistema por parte del usuario y el soporte continuo al usuario que incluye asistencia técnica, consultas con el usuario y registro de las peticiones de soporte en el **Histórico de Peticiones de Soporte**. Así, este proceso puede desencadenar la actividad del Proceso de Mantenimiento que provee información re-entrante al ciclo de vida del software.

Actividades a realizar:

- Operar el sistema.
- Proveer de asistencia técnica y consultas.
- Mantener el histórico de peticiones de soporte.

Documentos de salida:

- Histórico de peticiones de soporte.

PROCESO DE MANTENIMIENTO

Se interesa por los errores, defectos, fallos, mejoras y cambios del software. Un requisito de mantenimiento del software inicia los cambios del ciclo de vida del software; éste se reasigna y ejecuta.

El mantenimiento se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas por la evolución del entorno del software y a las modificaciones debidas a los cambios de los requisitos del cliente dirigidos a reforzar o a ampliar el sistema. El proceso de mantenimiento vuelve a aplicar los pasos del ciclo de vida, pero en el contexto del software ya existente; de este modo, se considera el Proceso de Mantenimiento como iteraciones de desarrollo.

Durante el mantenimiento se encuentran tres tipos de cambios:

- Corrección: incluso llevando a cabo las mejores actividades de garantía de calidad, es muy probable que el cliente descubra defectos en el software. El mantenimiento correctivo cambia el software para corregir los defectos.
- Adaptación: con el paso del tiempo es probable que cambie el entorno original (por ejemplo, la Unidad Central de Proceso, el Sistema Operativo, los periféricos) para el que se desarrolló el software. El mantenimiento adaptativo consiste en modificar el software para acomodarlo a los cambios de su entorno externo.
- Mejora: conforme utilice el software, el cliente/usuario puede descubrir funciones adicionales que podría interesar que estuvieran incorporadas en el software. El mantenimiento perfectivo amplía el software más allá de sus requisitos funcionales originales.

La salida de este proceso son **Recomendaciones de Mantenimiento** que entran al ciclo de vida del software en el Proceso de Exploración de Conceptos para mejorar la calidad del sistema software.

Actividad a realizar:

- Reaplicar el ciclo de vida del software.

Documentos de salida:

- Orden de mantenimiento.
- Recomendaciones de mantenimiento.

PROCESO DE RETIRO

Es la jubilación de un sistema existente de su soporte activo o de su uso mediante el cese de su operación o soporte, o mediante su reemplazamiento tanto por un nuevo sistema como por una versión actualizada del sistema existente. Si el sistema en uso, sea manual o automatizado, se reemplaza por un nuevo sistema se requiere un período de operación dual, denominado ensayo en paralelo. En este período se utiliza el sistema en retiro para los resultados oficiales, mientras se completa la preparación del nuevo sistema para la operación formal. Es un período de formación del usuario sobre el nuevo sistema y de validación del mismo.

Actividades a realizar:

- Notificar al usuario.
- Conducir operaciones en paralelo (si se aplica).
- Retirar el sistema.

Documentos de salida:

- Plan de retiro.

4.8. PROCESOS INTEGRALES DEL PROYECTO

Son procesos simultáneos y complementarios a los procesos orientados al desarrollo. Incluyen actividades imprescindibles para que el sistema construido sea fiable (procesos de verificación y validación, gestión de la configuración) y sea utilizado al máximo de sus capacidades (procesos de formación, documentación).

Los Procesos Integrales comprenden dos tipos de actividades:

- aquellas que se realizan discretamente y se aplican dentro de un ciclo de vida del software, y
- aquellas que se realizan para completar otra actividad. Estas son actividades que se invocan (llamadas como a una subrutina) y no se aplican dentro del modelo de ciclo de vida del software para cada instancia.

PROCESO DE VERIFICACIÓN Y VALIDACIÓN

Abarca la planificación y la realización de todas las tareas de verificación, incluyendo pruebas de verificación, revisiones y auditorías, y todas las tareas de validación, incluyendo pruebas de validación, que se ejecutan durante el ciclo de vida del software para asegurar que se satisfacen todos los requisitos del software.

Una actividad útil para la verificación y la validación del software es la prueba del

software. Constituye el proceso de ejecución del software con determinados datos de entrada, denominados juego de ensayo, para observar los resultados que produce y compararlos con los resultados que teóricamente (de acuerdo con las especificaciones) debería producir, para esos datos de entrada, con el objeto de detectar posibles fallos. Las pruebas del software sólo podrán realizarse cuando, en el proceso de desarrollo, ya exista código ejecutable.

La depuración es un proceso frecuentemente asociado a las pruebas (y a algunas otras actividades de verificación y validación) que consiste en tratar de deducir dónde están los defectos en el software que provocan que éste no funcione adecuadamente. Estudia los resultados de pruebas y otras actividades de control para intentar buscar qué está mal en el software.

Este proceso se aplica en cada proceso y producto del ciclo de vida del software.

Actividades a realizar:

- Planificar la verificación y validación.
- Ejecutar las tareas de verificación y validación.
- Recoger y analizar los datos de las métricas
- Planificar las pruebas.
- Desarrollar las especificaciones de las pruebas.
- Ejecutar las pruebas.

Documentos de salida:

- Plan de verificación y validación.
- Informes de evaluación.
- Plan de pruebas.
- Especificación de las pruebas.
- Informe resumen de las pruebas.
- Software probado.

Técnicas a utilizar:

- Técnicas de Prueba de Caja Blanca.
    - Cobertura de Sentencias.
    - Cobertura de Decisión o de Ramificación.
    - Cobertura de Condición.
    - Cobertura de Decisión/Condición.
    - Cobertura de Condición Múltiple.

- Técnicas de Prueba de Caja Negra.

        - Partición de Equivalencias.
        - Análisis de Valores Límites.
        - Gráficos de Causa-Efecto.
        - Conjetura de Errores.
    - Revisiones Formales.
    - Auditorías.

PROCESO DE GESTIÓN DE LA CONFIGURACIÓN

Este proceso involucra un conjunto de actividades desarrolladas para gestionar los cambios durante todo el ciclo de vida del software. Identifica la estructura de un sistema (qué rutinas, módulos, datos, ficheros, etc., lo componen) en un momento dado (incluso cuando se está desarrollando) a lo que se denomina **Configuración del Sistema**. Su objetivo es el control de los cambios en el sistema, mantener su coherencia y su "rastreabilidad" o "trazabilidad", y poder realizar auditorías de control sobre la evolución de las configuraciones.

La gestión de la configuración realiza las siguientes funciones:

- identificación de la configuración de un sistema: descripción documentada de las características reales del sistema en un determinado momento,
- control de la configuración: establece la configuración inicial o básica y controla los cambios en los elementos de la misma,
- informes del estado de la configuración, y
- auditorias de configuración: revisiones independientes de la configuración para comprobar que los elementos de la configuración cumplen los requisitos de configuración establecidos.

En resumen, la gestión de la configuración del software identifica los elementos de un proyecto de desarrollo del software (especificaciones, código, planes, etc.) y provee tanto el control de los elementos identificados como la generación de informes de estado de la configuración. Todo esto con el objeto de conseguir visibilidad en la gestión y responsabilidad de la misma durante el ciclo de vida del software.

Actividades a realizar:

- Planificar la gestión de la configuración.
- Realizar la identificación de la configuración.
- Realizar el control de la configuración.
- Realizar la información del estado de la configuración.

Documentos de salida:

- Plan de gestión de configuración del software.
- Orden de cambio de ingeniería.
- Cambio de estado.
- Informe de estado.

PROCESO DE DESARROLLO DE DOCUMENTACIÓN

El Proceso de Desarrollo de Documentación para el desarrollo y uso del software es el conjunto de actividades que planifican, diseñan, implementan, editan, producen, distribuyen y mantienen los documentos necesarios para los desarrolladores y los usuarios.

Actividades a realizar:

- Planificar la documentación.
- Implementar la documentación.
- Producir y distribuir la documentación.

Documentos de salida:

- Plan de documentación

PROCESO DE FORMACIÓN

Incluye la planificación, desarrollo, validación e implementación de los programas de formación de desarrolladores, personal de soporte técnico y clientes y la elaboración de los materiales de formación adecuados.

Para conseguir una utilización efectiva del sistema software, se debe proporcionar a los usuarios del sistema instrucciones, guía y ayuda para el entendimiento de las capacidades del sistema y de sus limitaciones. Por ello, es imprescindible la formación de los usuarios en el nuevo sistema.

El desarrollo de productos software de calidad depende en gran medida de los conocimientos de las personas y del personal especializado involucrados en el proyecto. Por ello, es esencial la formación de los desarrolladores y personal de soporte técnico.

Actividades a realizar:

- Planificar el programa de formación.
- Desarrollar los materiales de formación.
- Validar el programa de formación.
- Implementar el programa de formación.

Documentos de salida:

- Plan de formación.

5. MAPA DE ACTIVIDADES DE UN PROYECTO

Como ya se ha mencionado, no todos los proyectos de desarrollo de software son iguales. De hecho, ya se ha visto, que al inicio del proceso, un momento crítico es aquél de la decisión de qué ciclo de vida se elegirá para el proyecto en cuestión. Una vez que se ha hecho tal selección, y guiado en cierto modo por ella, se debe adaptar el proceso software genérico al modelo de ciclo de vida elegido. Es decir, establecer el mapa de actividades del proyecto. El proceso software visto en el apartado anterior es un proceso genérico, que obligatoriamente debe adaptarse para cada proceso. La adaptación se lleva a cabo precisamente mediante el establecimiento del mapa de actividades.

El mapa de actividades es una tabla donde se marcan qué actividades del proceso software genérico se van a ejecutar para un determinado proyecto. Existen desarrollados ya ciertos mapas de actividades para los tipos de proyectos más comunes (proyecto grande, proyecto pequeño, etc.), que pretenden facilitar la labor del jefe de proyecto. En cualquier caso, las tablas existentes se deben siempre comprobar y adaptar para un proyecto concreto. Hay que insistir en que no hay dos proyectos de desarrollo de software iguales.

El tipo de proyecto está muy estrechamente relacionado con el ciclo de vida. Por ejemplo, un proyecto pequeño casi siempre se corresponde con un ciclo de vida en cascada; un proyecto medio innovador, se suele corresponder con un ciclo de vida con prototipado; etc.

Por tanto, como puede verse en la Tabla 5.1., el mapa de actividades es una tabla de dos entradas. Una entrada es el proceso software con sus actividades y la otra entrada el ciclo de vida elegido descompuesto en sus etapas.

		Requisitos	Diseño	.....
Proceso software	Actividad 1 Actividad 2 . . . . Actividad n			

Figura 5.1. Mapa de Actividades

En el mapa de actividades simplemente se marcan con una cruz las actividades que se llevarán a cabo. Además, puede incluirse más información del tipo: importancia de la actividad, si está marcada con un + significa mucha y si está marcada con un - significa normal; tipo de actividad, obligatoria (marcada con una O) o condicional (marcada con una C).

A continuación, en la Tabla 5.1, aparece un ejemplo de mapa de actividades para un proyecto que ha elegido un ciclo de vida en cascada con nueve etapas definidas del siguiente modo:

- **Factibilidad (FA).** Definir el producto software, y determinar su factibilidad en el ciclo de vida y superioridad con respecto a productos alternativos.
- **Requisitos (RS).** Una completa especificación validada de las funciones requeridas, interfaces, y rendimiento para el producto software.
- **Diseño del Producto (DP).** Una completa especificación verificada de la arquitectura del hardware-software, estructura de control, y estructura de datos para el producto, junto con otros componentes necesarios como los manuales del usuario preliminar y los planes de prueba.
- **Diseño Detallado (DD).** Una completa especificación verificada de la estructura de control, estructura de datos, relaciones de interfaz, tamaño, algoritmos claves, y suposiciones de cada componente de programa.
- **Codificación (CO).** Un completo conjunto verificado de componentes del programa.
- **Integración (IN).** Un adecuado funcionamiento del producto software compuesto de los componentes software.
- **Implementación (IM).** Un sistema hardware-software funcionando operacionalmente a pleno, incluyendo tales objetivos como conversión del programa y datos, instalación, y entrenamiento.
- **Operación y Matenimiento (OM).** Un funcionamiento actualizado del sistema hardware-software. Este subobjetivo se repite para cada actualización.
- **Retiro (RE).** Una transición adecuada de las funciones realizadas para el producto y sus sucesores (si existe).

ACTIVIDADES DE LOS PROCESOS	FA	RS	DP	DD	CO	IN	IM	OM	RE
Proceso de Selección de un MCVS									
- Identificar los posibles MCVS	x								
- Seleccionar un modelo para el proyecto.	x								
Proceso de Iniciación, Planificación y Estimación del Proyecto									
- Establecer la matriz de actividades para el MCVS .	x								
- Asignar los recursos del proyecto.	x	x	x	x	x	x	x	x	x
- Definir el entorno del proyecto.	x	x		x					
- Planificar la gestión del proyecto.	x	x							
Proceso de Seguimiento y Control del Proyecto									
- Analizar riesgos.	x	x	x	x	x	x	x		
- Realizar la planificación de contingencias.		x	x	x	x	x	x		
- Gestionar el proyecto.	x	x	x	x	x	x	x	x	x
- Implementar el sistema de informes de problemas.	x	x	x	x	x	x	x	x	x
- Archivar registros.		x	x	x	x	x	x	x	x
Proceso de Gestión de Calidad del Software									
- Planificar la garantía de calidad del software.		x	x						
- Desarrollar métricas de calidad.		x	x	x					
- Gestionar la calidad del software.	x	x	x	x	x	x	x	x	x
- Identificar necesidades de mejora de la calidad.	x	x	x	x	x	x	x	x	x
Proceso de Exploración de Conceptos									
- Identificar las ideas o necesidades.	x								
- Formular las soluciones potenciales.	x	x							
- Dirigir los estudios de viabilidad.	x	x							
- Planificar la transición del sistema (si se aplica).	x	x							x
- Refinar y Finalizar la idea o necesidad.		x							
Proceso de Asignación del Sistema									
- Analizar las funciones del sistema.		x	x						
- Desarrollar la arquitectura del sistema.		x	x						
- Descomponer los requisitos del sistema.		x							
Proceso de Análisis de Requisitos									
- Definir y Desarrollar los requisitos del software.		x	x						
- Definir los requisitos de interfaz.		x	x						
- Priorizar e Integrar los requisitos del software.		x	x						

ACTIVIDADES DE LOS PROCESOS	FA	RS	DP	DD	CO	IN	IM	OM	RE
Proceso de Diseño									
- Realizar el diseño preliminar.			x						
- Analizar el flujo de información.			x	x					
- Diseñar la base de datos (si se aplica).			x	x					
- Diseñar las interfaces.			x	x					
- Seleccionar o Desarrollar algoritmos (si se aplica).		x		x					
- Realizar el diseño detallado.				x					
Proceso de Implementación e Integración									
- Crear los datos de prueba.			x	x					
- Crear el código fuente.			x	x					
- Generar el código objeto.			x	x					
- Crear la documentación de operación.			x	x					
- Planificar la integración.			x						
- Realizar la integración.				x					
Proceso de Instalación y Aceptación									
- Planificar la instalación.						x			
- Distribuir el software.							x		
- Instalar el software.							x		
- Cargar la base de datos (si se aplica).							x		
- Aceptar el software en el entorno de operación.							x		
- Realizar las actualizaciones.								x	
Proceso de Operación y Soporte									
- Operar el sistema.								x	
- Proveer de asistencia técnica y consultas.								x	
- Mantener el histórico de peticiones de soporte.								x	
Proceso de Mantenimiento									
- Realizar el mantenimiento correctivo.								x	
- Reaplicar el ciclo de vida del software.								x	
Proceso de Retiro									
- Notificar al usuario.								x	x
- Conducir operaciones en paralelo (si se aplica).									x
- Retirar el sistema.									x
Proceso de Verificación y Validación									
- Planificar la verificación y validación.		x	x						
- Ejecutar las tareas de verificación y validación.		x	x	x	x	x	x	x	x
- Recoger y Analizar los datos de las métricas.		x	x	x	x	x	x	x	x
- Planificar las pruebas.				x	x				
- Desarrollar las especificaciones de las pruebas.				x	x				
- Ejecutar las pruebas.					x	x	x		

ACTIVIDADES DE LOS PROCESOS	FA	RS	DP	DD	CO	IN	IM	OM	RE
Proceso de Configuración									
- Planificar la gestión de configuración.		x	x						
- Realizar la identificación de la configuración.		x	x	x	x	x			
- Realizar el control de la configuración.			x	x	x	x	x	x	x
- Realizar la información del estado de la configuración.			x	x	x	x	x	x	x
Proceso de Documentación									
- Planificar la documentación.		x	x						
- Implementar la documentación.				x	x				
- Producir y Distribuir la documentación.						x	x		
Proceso de Formación									
- Planificar el programa de formación.		x	x						
- Desarrollar los materiales de formación.			x	x	x	x			
- Validar el programa de formación.						x	x		
- Implementar el programa de formación.							x		

Tabla 5.2. Matriz de Actividades para un ciclo de vida en cascada con nueve etapas

Como puede verse en la Tabla 5.2, el mapa marca qué actividades del Proceso Software deberán realizarse en cada una de las etapas del Ciclo de Vida. Existen actividades que es necesario realizarlas una única vez (por ejemplo, la selección de un modelo de ciclo de vida). Sin embargo, existen otras actividades que se realizan en cada una de las etapas (por ejemplo, gestionar el proyecto). Obviamente, dependiendo del ciclo de vida elegido el mapa de actividades variará.

Por tanto, el mapa de actividades es el primer paso para conseguir una organización del proyecto que lleve a su gestión. A partir del mapa, puede pasarse a una estimación del tiempo y costo de cada una de las actividades, y por tanto, del proyecto global; a una asignación de recursos para cada actividad, etc.

6. BIBLIOGRAFÍA

BIBLIOGRAFÍA BÁSICA

[Böehm, 1976] Böehm, B. **Software Engineering**. IEEE Trans. Computers C-25, 12. Dec. 1976.

[Böehm, 1986] Böehm, B. W. **A Spiral Model of Software Development and Enhancement**. ACM Software Engineering Notes 11, 4. 1986.

[Böehm, 1988] Böehm, B. W. **A Spiral Model of Software Development and Enhancement**. Computer, May 1988, pp. 61-72.

[Budde, 1984] Budde, R., K. Kuhlenkamp, L. Mathiassen, and H. Zullighoven. **Approaches to Prototyping**. Springer-Verlag, New York 1984.

[Humphrey, 1995] Humphrey, W. S. **A discipline for Software Engineering**. Addison-Wesley. Readings, Massachusetts, EE.UU. 1995.

[IEEE, 1989] IEEE Std. 1074-1989. IEEE Standard Software Life Cycle Processes. 1989.

[McCracken, 1982] McCracken, D.D., and Jackson, M.A. **Life-Cycle Concept Considered Harmful**. ACM SW Eng. Nates, Apr. 1982, pp. 29-32.

[Pressman, 1993] Pressman, R. S. **Ingeniería del Software: Un enfoque práctico**. 3 ed. McGraw-Hill, Madrid. 1993.

[Royce, 1970] Royce, W. W. **Managing the Development of Large Software Systems**. Proc. 9th. Intern. Conf. Software Engineering. IEEE Computer Society, 1987, 328-338. Originally published in Proc. WES-CON, 1970.

BIBLIOGRAFÍA OPTATIVA

[Balzer, 1981] Balzer, R. **Transformational Implementation: An Example**. IEEE Trans. Software Eng. SE-7, 1. 1981.
Para el ciclo de vida con prototipado.

[Balzer, 1982] Balzer, R., N. Goldman, and D. Wile. **Operational Specifications as the Basis for Rapid Prototyping**.
ACM Software Engineering Notes 7,5. 1982.
Para especificaciones operativas.

[Balzer, 1983 (a)] Balzer, R., D. Cohen, M. Feather, N. Goldman, W. Swartout and D. Wile. **Operational Specifications as the Basis for Specification Validation**. In Theory and Practice of Software Technology, Ferrari, Bolognani, and Goguen, eds. North-Holland, 1983.
Para especificaciones operativas.

[Basili, 1975] Basili, V.R., and A.J. Tumer. **Iterative Enhancement: A Practical Technique for Software Development.** IEEE Trans. Software Eng.SE-1, 4. Dec. 1975.
Para el modelo de ciclo de vida de refinamiento sucesivo.

[Bauer, 1976] Bauer, F. L. **Programming as an Evolutionary Process.** Proc. 2nd. Intern. Conf. Software Engineering. IEEE Computer Society, Jan. 1976.
Para especificaciones operativas.
Para ciclo de vida de transformación continua.

[Biggerstaff, 1984] **Special Issues on Software Reusability.** T. Biggerstaff and A. Perlis, eds. IEEE Trans. Software Eng. SE-10, 5. Sept. 1984.
Para el modelo de ciclo de vida por ensamblaje de componentes reutilizables.

[Böehm, 1984] Böehm, B. W., T. Gray, and T. Seewaldt. **Prototyping vs. Specifying: A Multi-project Experiment.** Proc. 7th. Intern. Conf. Soft. Engr. 1984.
Variación del modelo de ciclo de vida en cascada.
Para el ciclo de vida con prototipado.

[Curtis, 1987] Curtis, B., H. Krasner, V. Shen, and N. Iscoe. **On Building Software Process Models Under the Lamppost.** Proc. 9th. Intern. Conf. Software Engineering. IEEE Computer Society, April 1987.
Potencialidades y limitaciones de la automatización del proceso software.

[Deming, 1982] Dening, W.E. **Out of the crisis.** Center for Advanced Engineering. Massachussets Institute of Technology. Cambridge, Massachusetts, EE.UU. 1982.
Sobre la definición del proceso software en una organización.

[Distaso, 1980] Distaso, J. **Software Management - A Survey of Practice in 1980.** Proceedings IEEE 68, 9. 1980.
Variación del modelo de ciclo de vida en cascada.

[Goguen, 1986] Goguen, J. **Reusing and Interconnecting Software Components.** Computer 19, 2. Feb. 1986.
Para el modelo de ciclo de vida por ensamblaje de componentes reutilizables.

[Hekmatpour, 1987] Hekmatpour, S. **Experience with Evolutionary prototyping in a Large Software Project.** ACM Software Engineering Notes 12, 1. 1987.
Para el ciclo de vida con prototipado.

[Hoffnagel, 1975] Hoffnagel, G. F., and W. Beregi. **Automating the Software Development Process.** IBM Systems Journal. 24, 2. 1985.
Potencialidades y limitaciones de la automatización del proceso software.

[Huseth, 1986] Huseth, S., and D. Vines. **Describing the Software Process.** Proc. 3rd. Intern. Software Process Workshop. IEEE Computer Society, 1986.
Potencialidades y limitaciones de la automatización del proceso software.

[Lehman, 1984 (a)] Lehman, M. M., V. Stenning, and W. Turski. **Another Look at Software Development Methodology.** ACM Software Engineering Notes 9, 2. April 1984.
Para modelo de ciclo de vida visto como una transformación continua.

[Lehman, 1984 (b)] Lehman, M. M. **A Further Model of Coherent Programming Processes.** Proc. Software Process Workshop. IEEE Computer Society, 1984.
Para modelo de ciclo de vida visto como una transformación continua.

[MIL-STD-2167, 1987] Dept. of Defense. DRAFT Military Standard: Defense System Software Development. DOD-STD-2167A.
Estándar para el desarrollo de software para el Departamento de Defensa estadounidense.

[Neighbors, 1984] Neighbors, J. **The Draco Approach to Constructing Software from Reusable Components.** IEEE Trans. Software Eng. SE-10, 5. Sept. 1984.
Para el modelo de ciclo de vida por ensamblaje de componentes reutilizables.

[Osterweil, 1987] Osterweil, L. Software Processes are Software Too. Proc. 9th. Intern. Conf. Software Engineering. IEEE Computer Society. April 1987.
Potencialidades y limitaciones de la automatización del proceso software.

[Paulk, 1993] Paulk, M. C., Curtis, B. Chrisis. M. B. **Capability Maturiry Model for Software. Versión 1.1.** Software Engineering Institute Technical Report CMU/SEI-93-TR. Pittsburgh, Pensylvania, EE.UU. 1993.
Descripción del Modelo de Madurez del SEI.

[Scacchi, 1984] Scacchi, W. **Managing Software Engineering Projects: A Social Analysis.** IEEE Trans. Software Eng. SE-10, 1. Jan, 1984.
Variación del modelo de ciclo de vida en cascada.

[[Sen, 1982] Special Issue on Rapid Protityping. ACM Software Engineering Notes 7, 5. Dec. 1982.
Para variaciones sobre estrategias de prototipado, demostración, reutilización, etc.

[Tully, 1984] Tully, C. **Software Development Models**. Proc. Software Process Workshop. IEEE Computer Society, 1984.
Para el modelo de ciclo de vida con emisión gradual.

[Wirth, 1971] Wirth, N. **Program Development by Stepwise Refinement**. Comm. ACM 14, 4. April 1971.
Para el modelo de ciclo de vida de refinamiento sucesivo.

[Zave, 1984] Zave, P. **The Operational Versus the Conventional Approach to Software Development**. Comm. ACM 27. Feb. 1984.
Para especificaciones operativas.

7. Trabajo práctico

Dinámica de trabajo:

- A continuación se presentan 9 artículos relatando un caso que la prensa denominó "El Robot Asesino".
- Cada grupo deberá tomar un articulo (en lo posible no deberán tomar dos grupos el mismo articulo).
- Cada grupo deberá analizar el articulo seleccionado y responder a las preguntas citada a continuación:
    1. Identifique los errores de Gestión de Proyecto.
    2. ¿Estaban las pruebas diseñadas correctamente?.¿Quién es el responsable del diseño de las pruebas?. Justifique.
    3. Identifique los errores en la aplicación de la metodología para Análisis y Diseño de Sistemas de Información en relación a los requerimientos y el diseño lógico.
    4. Identifique los errores de codificación.
    5. Enumere las causas que generaron el accidente. Seleccione dos causas que a su criterio sean las más relevantes. Justifique su selección.
    6. Identifique los aspectos éticos que fueron vulnerados en el caso analizado.
- Una vez respondidas las preguntas cada grupo deberá enviar por correo electrónico al docente un documento Word en que se indique:
    - Nombre de los integrantes del grupo,
    - Articulo seleccionado, y
    - Respuestas a las preguntas del cuestionario.
- En la clase siguiente cada grupo tendrá 10 minutos para exponer lo analizado.

7.1. INTRODUCCION

El caso del Robot Asesino es una historia que combina elementos de Ingeniería del Software y de ética en la computación.

Los artículos de esta historia comienzan con la acusación por homicidio no premeditado a un programador. Este programador había escrito un código defectuoso que causó la muerte de un operador de robot, lentamente. También se presentan, a lo largo de varios artículos, factores de la corporación que también contribuyeron en el accidente. Se muestra el desarrollo del software como un proceso social, presentando

varios aspectos de la compleja tarea de construir software para un mundo real, y algunos temas éticos relacionados con esta complejidad.

El texto consta de una introducción y 9 artículos:

- Los involucrados: En la cual se caracterizan los personajes de la historia.
- Articulo 1: Programador de Silicon Valley acusado por homicidio no premeditado.
- Articulo 2: Los que desarrollaron el "Robot Asesino" trabajaron bajo una enorme presión.
- Articulo 3: Los compañeros acusan: el programador del "Robot Asesino" era una estrella.
- Articulo 4: El proyecto del "Robot Asesino" controvertido tema desde el vamos.
- Articulo 5: Silicon Techtronics prometió entregar un robot seguro.
- Articulo 6: La interfaz del "Robot Asesino".
- Articulo 7: Ingeniero del software cuestiona la autenticidad de las pruebas del software del "Robot Asesino".
- Articulo 8: Empleado de Silicon Techtronics admite la falsificación de las pruebas del software.
- Articulo 9: Conversación con el Dr. Harry Yoder.

Vale aclara que este punto y el texto donde se caracterizan los personajes de la historia no se deben utilizar como texto de análisis.

Este artículo fue utilizado por Richard G. Epstein. "El uso de escenarios sobre ética de la computación en la educación de Ingeniería del Software: el caso del robot asesino." Enseñanza de Ingeniería de Software: Anticipo de la 7ma. Conferencia SEI CSEE, San Antonio. Jorge L. Díaz-Herrera, editor. Notas del discurso en Computer Science 750. Springer-Verlag 1994.

7.2. LOS INVOLUCRADOS

El caso del robot asesino consta de nueve artículos publicados en periódicos, un artículo de un jornal y de una entrevista publicada en una revista. Esta historia intenta llamar la atención sobre tópicos de ética en la computación y en la ingeniería de software.

Las personas e instituciones involucradas en esta historia son enteramente ficticias (excepto por las referencias a las Universidades de Carnegie-Mellon y Purdeue y a los venerables computadores científicos Ben Shneiderman y Jim Foie). Se eligió Silicon

Valley para la ubicación del accidente debido a que éste es un icono de la alta tecnología. Todas las personas e instituciones nombradas en Silicon Valley son ficticias.

La caracterización de los personajes:

- **Alex Allendale**, Abogado, contratado para defender a Randy Samuels.
- **Jan Anderson**, programadora y analista de Silicon Techtronics. Se oponía al uso del modelo de cascada en el proyecto del robot y fue despedida por ser honesta.
- **Turina Babbage**, presidente de la Association for Computing Machinery (ACM). Anuncia que ACM realizaría una investigación sobre violaciones al Código de ética de la ACM por parte de los empleados de Silicon Techtronics.
- **Robert Franklin**, periodista del SENTINEL-OBSERVER de Silicon-Valley. Entrevistó al Profesor Harry Yoder para conocer la visión de un experto en ética sobre la evolución del caso del robot asesino. La entrevista fue publicada en la revista dominical del SENTINEL- OBSERVER.
- **Horace Gritty**, Profesor de Ciencias de la Computación.
- **Sandra Herdenson**, estudiante reciba en la Universidad de Silicon Valley. Coperó en la investigación sobre procedimientos de aseguramiento de la calidad en la Universidad de Silicon Valley.
- **Ray Jonson**, Jefe de División Robótica en Silicon Techtronics. La División Robótica necesitaba un robot exitoso.
- **Marta**, fuente anónima de un periódico. Una persona interna de Silicon Valley que dio al SENTINELA-OBSERVER información sobre la dinámica de grupo del proyecto del robot Robbie CX30.
- **Bart Matthews**, operador del robot. Un programa de computación con defectos causo que el robot Robbie CX30 lo matara.
- **Roberta Matthews**, viuda de Bart.
- **Jane McMurdock**, Fiscal de la Ciudad de Silicon Valley. Fue quien incrimina a Randy Samuels con los cargos de asesinato no premeditado.
- **Mabel Muckraker**, periodista del Silicon Valley Sentinel Observer. Fue designada a la historia del robot asesino por su reputación de eficaz reportera investigadora.
- **Bill Park**, profesor de física de la Universidad de Silicon Valley. Confirmo que Randy Samuels malinterpreto las ecuaciones de la dinámica del robot.
- **Randy Samuels**, programador. Escribió el código del programa que causó que el robot Robbie CX30 oscilara con gran amplitud, matando a su operador, Bart Matthews.

- **Sam Reynolds**, Gerente del proyecto del CX30, Ray Johson era su superior inmediato. SU experiencia fue adquirida en el campo del procesamiento de datos, pero fue puesto al mando del proyecto CX30, muy a su pesar. Estaba a favor de aplicar el modelo de cascada del desarrollo del software.
- **Robbie CX30**, el robot. Robbie jamás tuvo un mal pensamiento hacia nadie, aún así se torno un salvaje asesino.
- **Wesley Silber**, profesor de Ingeniería del Software en la Universidad del Silicon Valley. Condujo una revisión e los procedimientos de aseguramiento de la calidad en Silicon Techtronics.
- **Sharon Skinner**, profesora de psicología en la Universidad de Silicon Valley. Veía a Randy Samuels como una persona aplicada a su tarea, susceptible por demás a las criticas.
- **Valeria Thomas**, abogada, contratada por Sam Reynolds.
- **Michael Waterson**, presidente y presidente ejecutivo de Silicon Techtronics. Puso a Sam Reynolds a cargo del proyecto Robbie CX30 como una medida para recortar gastos. Contribuyo generosamente a la campaña de reelección de James McMurdock. Contrato al Dr. Silber para llevar adelante una investigación sobre aseguramiento de calidad en Silicon Techtronics.
- **Max Worthington**, jefe de seguridad de Silicon Techtronics. Monitoreaba la comunicación electrónica entre los empleados y así descubrió a Cindy Yardley.
- **Ruth Whiterspoon**, analista y programadora vocera del comité "Justicia para Randy Samuels". Defendía a Randy Samuels sobre la base que Silicon Techtronics estaba legalmente obligada a entregar un robot confiable y seguro.
- **Cindy Yardley**, empleada y probadora de software de Silicon Techtronics. Admitió la falsificación de las pruebas de software con el fin de resguardar el puesto de trabajo de sus compañeros.
- **Harry Yoder**, profesor de tecnología de computación y ética. Examino, en una entrevista publicada en la revista dominical del SENTINEL – OBSERVER, la tensión entre las responsabilidades individuales y corporativas.

7.3. LOS ARTICULOS

7.3.1. Articulo 1- Programador de Silicon Valley acusado por homicidio no premeditado. El error del programa causó la muerte del operador del robot

Especial para el SENTINEL – OBSERVER de Silicon Valley.

Jane McMurdock, fiscal de la ciudad de Silicon Valley, anunció en la fecha la acusación de Randy Samuels con los cargos de asesinato no premeditado. Samuels trabajaba

como programador en Silicon Techtronics, Inc., una de las empresas mas nuevas de Silicon Valley en la arena de la alta tecnología. El cargo involucra la muerte de Bart Matthews, quien fuera muerto el pasado mes de mayo por un robot de la línea de armado.

Matthews, quien trabajaba como operador de robot en Cybernetics, Inc., en Silicon Heights, fue aplastado y murió como consecuencia de ello, cuando el robot que estaba operando produjo un malfuncionamiento y comenzó a oscilar su "brazo" violentamente. El brazo del robot alcanzó a Matthews, arrojándolo contra una pared y aplastando su cráneo. Matthews murió en forma casi instantánea en un caso que conmocionó e indignó a muchos en Silicon Valley. De acuerdo con el dictamen de cargos, Samuels fue quien escribió la pieza del programa de computadora en particular, que fue la responsable de la falla del robot. "Hay una evidencia incriminatoria", anunció triunfante, McMurdock en una conferencia de prensa mantenida en la Corte.

"Tenemos la fórmula manuscrita, suministrada por el físico del proyecto, que se suponía que tenía que programar Samuels. Pero negligentemente malinterpretó la formula, y esto llevó a una horrible muerte. La sociedad debe protegerse de los programadores que comenten errores descuidadamente o de lo contrario nadie estará a salvo, y menos que nadie nuestra familia e hijos", dijo:

- El SENTINEL – OBSERVER ha podido obtener una copia de la muestra de la fórmula manuscrita en cuestión. En realidad, existen tres fórmulas similares, garabateadas en un papel amarillo de un block borrador tamaño oficio. Cada una de las fórmulas describe el movimiento del brazo del robot en una dirección: este – oeste, norte- sur y arriba – abajo.
- El SENTINEL – OBSERVER mostró entonces las fórmulas a Bill Park, profesor de física de la Universidad de Silicon Valley. Este confirmó que estas ecuaciones podían ser usadas para describir el movimiento del brazo del robot.
- El SENTINEL – OBSERVER mostró entonces el código del programa a Bill Park, escrito por el acusado en lenguaje C de programación. Preguntamos al Profesor Park, quien está muy familiarizado con este y muchos otros lenguajes de programación, si el código era o no correcto para las formulas dadas del brazo del robot.

  La respuesta del Profesor Park fue inmediata: "¡Por Júpiter!". Parece que interpretó los puntos y de las fórmulas como barras y, e hizo lo mismo con las x y las z. Se suponía que tenia que usar derivadas, pero en su lugar tomo los promedios!. Si me preguntan, ¡es culpable como el mismo demonio!".
- El SENTINEL – OBSERVER no pudo contactar a Samuels para entrevistarlo. "Se encuentra profundamente deprimido por esto", nos dijo la novia por teléfono. "Pero, Randy cree que va a aliviarse en cuanto pueda decir su versión de la historia.".

7.3.2. Articulo 2 - Los que desarrollaron al "Robot Asesino" trabajaron bajo una enorme presión

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

El SENTINEL – OBSERVER tomó conocimiento hoy que Randy Samuels y otros que trabajaron en el proyecto del "robot asesino" en Silicon Techtronics, estuvieron bajo tremendas tensiones para finalizar el software del robot para el 1° de enero de este año. Según una fuente bien informada, los altos niveles gerenciales advirtieron a los integrantes del staff del proyecto que "rodarían cabezas" si no se cumplía con el objetivo del 1° de enero.

Randy Samuels, programador de Silicon Techtronics, fue acusado la semana pasada bajo cargo de asesinato no premeditado en el ahora famoso caso del "Robot asesino". Samuels escribió el software defectuoso que causó que el robot industrial de Silicon Techtronics, Robbie CX30, aplastara y lesionara fatalmente al operador, Bart Matthews. Matthews era un operador de robot de Cybernetics, Inc. Conforme a la fiscal de Silicon Valley, Jane McMurdock, Samuels malinterpretó la formula matemática, "volviéndolo al inofensivo Robbie un salvaje asesino".

Nuestra fuente informada, quien desea mantenerse en el anonimato y a quien llamaremos "Marta" por el resto de este artículo, tiene un íntimo conocimiento de todos los aspectos del proyecto Robbie CX30. Marta dijo al SENTINEL – OBSERVER que existía una enorme fricción entre el jefe de División de Robótica, Ray Jonson y el gerente del proyecto de Robbie CX30, Sam Reynolds. "Se odiaban a muerte" manifestó Marta al SENTINEL – OBSERVER en una entrevista exclusiva.

"Hacia junio del año pasado el proyecto se encontraba atrasado seis meses y Johnson se puso furioso. Había rumores que echarían a toda la División de Robótica, que él lideraba, si Robbie (el robot CX30) no daba muestras de ser un éxito comercial. Él (Jonson) llamo a Sam (Reynolds) a su oficina y realmente lo destruyó. Quiero decir, uno podía oír los gritos desde el fondo de la oficina. Jonson le dijo a Sam que terminara el proyecto para el 1° de enero o de lo contrario "rodarían cabezas".

"Yo no estoy diciendo que Jonson le ordenara a Sam acortar camino", agrego Marta. "Creo que la idea de cortar camino estaba implícita. El mensaje fue: "acortá camino si querés mantener tu puesto".

De acuerdo con documentos provistos por Marta al SENTINEL – OBSERVER, el 12 de julio del año pasado fueron agregados al proyecto Robbie CX30 veinte nuevos programadores. Esto ocurrió días después de la tomentosa reunión entre Jonson y Reynolds que Marta contó.

De acuerdo a Marta, los nuevos contratados eran un desastre. "Jonson, unilateralmente, hizo los arreglos de estas contrataciones, seguramente desviando recursos de otros aspectos del proyecto Robbie (CX30). Reynolds se oponía con vehemencia a esto. Jonson solo conocía acerca de la fabricación del hardware. Esa era su especialidad. No pudo haber entendido de las dificultades que nosotros estamos teniendo con el software de la robótica. Usted no puede acelerar un proyecto de software agregando mas gente. No es como una línea de montaje".

Según Marta y otras fuentes, la contratación de estos nuevos veinte programadores llevó a que se hiciera una reunión entre Johson, Reynolds y todos los integrantes del proyecto de software de Robbie CX30. "Esta vez fue Sam (Reynolds) el que se puso furioso. Se quejó de que el proyecto no necesitaba más gente. Sostuvo que el problema principal era que Jonson y otros miembros a nivel directivo no entendían que el Robbie CX30 era fundamentalmente diferente a otras versiones anteriores del robot". Estas fuentes dijeron al SENTINEL – OBSERVER que los nuevos empleados no estaban totalmente integrados al proyecto, aún seis meses después de su ingreso, cuando diez robots Robbie CX30, incluido el robot que mato a Bart Matthews, ya habían sido despachados. Según Marta, "Sam solo quería mantener las cosas lo mas simples posible. No quería que el nuevo personal complicara las cosas". "Se pasaron seis meses leyendo manuales. La mayoría de los empleados nuevos no sabia nada de robots y Sam no estaba como para perder tiempo tratando de enseñarles". Según Marta, la reunión del 12 de junio se hizo famosa en la corporación Silicon Techtronics porque fue en esa reunión donde Ray Jonson anuncio su "Teoría Ivory Snow ["no existe el blanco perfecto, o bien, no hay más blanco que el blanco nieve"] de diseño y desarrollo de software". De acuerdo a Marta, "Ray (Johson) nos dio una gran presentación en multimedia, con diapositivas y todo. La esencia de esta "Teoría Ivory Snow" es simplemente que el blanco nieve es 99,44 % puro y que no hay razón por la que el software de robótica deba ser más puro que esto. Dijo repetidas veces que "El software perfecto era un oximoron".

Marta y otros personajes anónimos que se acercaron con información, retrataron a Jonson como un gerente con una desesperada necesidad de ser ayudado por un éxito en el proyecto. Versiones anteriores de Robbie, CX10 y CX20, fueron experimentales por naturaleza y nadie esperaba que fueran éxitos comerciales. De hecho, la División Robótica de Silicon Techtronis estaba operando con sus finanzas en rojo desde su concepción 6 años atrás. O triunfaba el CX30 o Silicon Techtronics quedaría fuera del negocio de robótica industrial. "Los robots Robbie anteriores tuvieron mucha prensa, especialmente aquí en Silicon Valley", dijo otra fuente que también quiere permanecer anónima. "Robbie CX30 iba a capitalizarse con la buena publicidad generada por los proyectos anteriores. Lo único es que Robbie CX30 era mas revolucionario de lo que Jonson quería admitir. CX30 representaba un paso gigante hacia delante en términos de sofisticación. Había muchísimas preguntas acerca de los parámetros industriales en

los que debería trabajar el CX30. Mucho de lo que debía ejecutarse era completamente nuevo, pero Jonson nunca lo pudo entender. Él solo nos veía como unos perfeccionistas. Uno de sus dichos favoritos era 'La perfección es enemiga de lo bueno'".

7.3.3. Articulo 3 - Los compañeros acusan: "El programador del `Robot Asesino´ era una estrella"

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

Randy Samuels, el que fuera el programador de Silicon Techtronics que fue acusado por escribir el software que causó el horrible incidente del "robot asesino" el pasado mes de mayo, era aparentemente una "prima donna" que encontraba muy difícil aceptar criticas, aseguraron hoy varios compañeros de trabajo.

En una rueda de prensa con varios compañeros de trabajo de Samuels en el proyecto del "robot asesino", el SENTINEL – OBSERVER pudo obtener importantes revelaciones acerca de la psiquis del hombre que pudo haber sido criminalmente responsable de la muerte de Bart Matthews, operador de robot y padre de 3 criaturas.

Con el permiso de los entrevistados, el SENTINEL – OBSERVER permitió a la profesora Sharon Skinner del Departamento de Psicología de Software en la Universidad de Silicon Valley, escuchar una grabación de la entrevista. La profesora Skinner estudia la psicología de los programadores y otros factores psicológicos que tienen impacto en el proceso de desarrollo del software.

"Estaría de acuerdo con la mujer que lo llamó "prima donna"", explicó la profesora Skinner. "Este es un término utilizado para referirse a un programador que simplemente no puede aceptar las críticas, o más precisamente, no puede aceptar su propia falibilidad."

"Randy Samuels tiene lo que nosotros, psicólogos de programadores, llamamos una personalidad orientada hacia una tarea, lindando con una personalidad orientada hacia sí mismo. Le gusta poder completar cosas, pero su ego esta muy densamente involucrado en su trabajo. En el mundo de la programación esto se considera "no, no", agregó la profesora Skinner en su oficina tapizada en libros.

La profesora Skinner continuó explicando algunos hechos adicionales sobre equipos de programación y personalidades del programador. "Básicamente, hemos encontrado que un buen equipo de programación requiere de una mezcla de personalidades, incluyendo a una persona que esté orientada hacia la interacción, que saca una enorme satisfacción del hecho de trabajar con otra gente, alguien que pueda ayudar a

mantener la paz y a que las cosas se muevan en una dirección positiva. Muchos programadores están orientados hacia lo que es la tarea, y esto puede ser problemático si se tiene un equipo donde son todos de este modo".

Los compañeros de trabajo de Samuels se mostraron muy reticentes a culpar a alguien por el desastre del robot, pero cuando se los presionó para que comentaran la personalidad de Samuels y sus hábitos laborales, surgieron varios hechos importantes. Samuels trabajaba en un equipo formado aproximadamente por una docena de analistas, programadores y probadores de software. (Esto no incluye a 20 programadores que fueron incorporados posteriormente y que nunca llegaron a estar activamente involucrados en el desarrollo del software de la robótica). Si bien cada individuo del equipo poseía una especialidad, casi todos están comprometidos en todo el proceso del software del principio al fin.

"Sam Reynolds tenía un background en el procesamiento de datos. Dirigió unos cuantos proyectos de software de esta naturaleza", dijo uno de los integrantes del equipo, refiriéndose al gerente del proyecto Robbie CX30. "Pero su rol en el proyecto era mas que nada de líder. Asistía a todas las reuniones importantes y lo mantenía a Ray (Ray Jhonson jefe del Departamento de robótica) sobre nuestras espaldas lo mas posible". Sam Reynolds, como ya fuera informado en el SENTINEL – OBSERVER de ayer, se encontraba bajo una severa presión para lograr producir un robot Robbie CX30 operativo para el 1° de enero de este año. Sam Reynolds no pudo ser ubicado para entrevistarlo ya sea sobre su rol en el incidente o sobre Samuels y sus hábitos de trabajo.

"Éramos un equipo democrático, a excepción del liderazgo provisto por Sam (Reynolds)", observó otro miembro del equipo. En el mundo del desarrollo del software, un equipo democrático es un equipo en donde todos los miembros de este tienen un decir en el proceso de toma de decisiones. "Desafortunadamente, nosotros éramos un equipo de individualistas muy ambiciosos, muy talentosos – si debo referirme a mi mismo – y muy opinadores. Randy (Samuels) era justo el peor del grupo. Lo que quiero decir es que teníamos, por ejemplo, a dos chicos y una chica con grados de maestría de la CMU1 , y no eran tan arrogantes como Randy."

Un compañero comentó sobre un incidente que Samuels causó en una reunión de aseguramiento de la calidad. Esta reunión involucraba a Samuels y a tres revisores de un módulo de software que Samuels había diseñado e implementado. Tales reuniones son llamadas "revisiones del código". Uno de los revisores menciono que Samuels había usado un algoritmo sumamente ineficiente (programa) para lograr un determinado resultado y Samuels "se puso todo colorado". Empezó a gritar una sarta de obscenidades y después se levantó y se fue. Y nunca regresó.

[1] CMU significa Universidad de Carnegie – Mellon, una líder nacional en enseñanza de Ingeniería del Software.

"Le enviamos un memo con un algoritmo mas rápido y a su tiempo usó este algoritmo en su módulo", agrego el colega.

El módulo de software del incidente de la reunión de aseguramiento de la calidad fue el primero en ser identificado como una falla en el "asesino" del operador de robot. No obstante, este colega se apuró a señalar que la eficacia del algoritmo no era un tópico en el malfuncionamiento del robot. Era solo que Randy hacia muy difícil para la gente el poderle comunicar las observaciones. Se tomaba todo muy a pecho. Se graduó con el puntaje más alto de la clase y luego se recibió con honores en Ingeniería del Software en Purdue. Definitivamente es muy inteligente".

"Randy, en su pared, tiene este inmenso cartel hecho en Banner", continuó este colega, "Decía: "DENME LA ESPECIFICACIÓN Y LES DARE UN PROGRAMA DE COMPUTACIÓN". Ese es el tipo de arrogancia que tenía y también demuestra que tenía muy poca paciencia para desarrollar y verificar las especificaciones. Amaba el aspecto de solucionar el problema, la programación propiamente dicha". No parecía que Randy Samuels quedó atrapado en el espíritu de la "programación sin egolatría", observó la profesora Skinner cuando escuchó esta parte de la entrevista con los colegas de trabajo de Samuels. "La idea de una programación sin egocentrismo es que el producto de software pertenece al equipo y no a los programadores individuales. La idea es estar abierto a las críticas y estar menos atado al trabajo propio. Ciertamente que la tarea de revisión de código es coherente con esta filosofía general". Una colega habló acerca de otro aspecto de la personalidad de Samuels: su capacidad de ayuda. "Randy odiaba las reuniones, pero era muy bueno con las relaciones uno a uno. Siempre estaba ansioso por ayudar. Recuerdo una vez que me encontraba encerrada en un camino sin salida y él, en vez de tan solo señalarme la dirección correcta, se hizo cargo del problema y lo resolvió el mismo. Se paso cerca de 5 días completos en mi problema". "Por supuesto que mirando en retrospectiva, hubiera sido mejor para el pobre Sr. Matthews y su familia que Randy se hubiese dedicado tan solo a sus propias cosas", agregó luego de una larga pausa.

7.3.4. Articulo 4 - El proyecto del "Robot Asesino" controvertido desde el vamos. Bandos enfrentados por el modo en que debía proseguir el proyecto

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

Dos grupos, comprometidos con diferentes filosofías de desarrollo de software, casi se enfrentan violentamente durante las reuniones iniciales de planeamiento para el Robbie CX30, el robot de Silicon Techtronics que mató a un obrero de la línea de ensamble el pasado mes de mayo. Estaba en cuestionamiento si el proyecto Robbie

CX30 debía proseguir de acuerdo con el "modelo de cascada" o el "modelo de prototipo".

El modelo de cascada y el prototipo son dos métodos comunes para organizar un proyecto de software. En el modelo de cascada, el proyecto de software pasa a través de etapas definidas de desarrollo. La primera etapa es la de análisis de requerimientos y especificaciones, durante la cual se intenta arribar a un acuerdo en cuanto a la funcionalidad detallada del sistema. A medida que el proyecto pasa de una etapa a la siguiente, existen limitadas oportunidades de dar marcha atrás y cambiar decisiones ya tomadas. Una desventaja es que los usuarios potenciales no tienen oportunidad de interactuar con el sistema hasta bien entrado el ciclo de vida del mismo.

En el modelo de prototipo, se pone un gran énfasis en producir un modelo de prototipo bien temprano durante el ciclo de vida del sistema. El prototipo es construido con el propósito de arribar a una especificación final de la funcionalidad del sistema propuesto. Los usuarios potenciales pueden interactuar con el prototipo en forma temprana y con frecuencia hasta que son acordados los requerimientos. Este enfoque le da los potenciales usuarios la oportunidad de interactuar con un sistema prototipo en forma temprana durante el ciclo de desarrollo y mucho antes que el sistema final esté diseñado y codificado.

En un memorando de fecha 11 de diciembre del anteaño pasado, Jan Anderson, miembro del equipo original del proyecto CX30, ataco duramente la decisión tomada por el gerente del proyecto Sam Reynolds de emplear el modelo en cascada. El SENTINEL – OBSERVER obtuvo una copia del memo de Anderson, dirigido a Reynolds, y Anderson verificó la autenticidad del memorando para este diario.
Reynolds despidió a Anderson el 24 de diciembre, justo dos semanas después de que ella escribiera el memo.

El memo de Anderson hace referencia a una reunión anterior en la que ocurrió un fuerte intercambio de opiniones relacionadas con la filosofía del desarrollo del software.

En el memo, Anderson subrayó el siguiente párrafo:

"No fueron mis intenciones impugnar su competencia durante la reunión de ayer, pero debo protestar con mi mayor vehemencia contra la idea de que completemos el software de Robbie CX30 siguiendo el modelo de cascada que Usted ya usó en otros proyectos. No necesito recordarle que aquellos eran proyectos de procesamiento de datos que involucraban el procesamiento de transacciones de negocio. El proyecto Robbie CX30 llevará un alto grado de interacción, tanto entre robot y componentes como entre robot y su operador. Dado que la interacción del operador con el robot es

tan importante, la interfaz no puede estar diseñada como una idea de último momento". Randy Samuels, a quien se lo acuso de asesinato no premeditado por la muerte de Bart Matthews, padre de 3 niños, había participado de la reunión del 11 de diciembre.

En una conservación con este diario, Anderson dijo que Samuels no tenía mucho que decir sobre la controversia cascada – prototipo, pero sí afirmó que daría "una mano" con tal de que exoneraran a Samuels.

"El proyecto fue sentenciado a muerte mucho antes de que Samuels malinterpretara las fórmulas", aclaró Anderson enfáticamente en la sala de su casa en los suburbios.
En conversación con este diario, Anderson hizo lo mejor de sí para explicar la controversia del método cascada vs. prototipo en términos simples. "El punto principal en realidad era si podíamos llegar a ponernos de acuerdo con los requerimientos del sistema sin dejar que los operadores del robot presintieran lo que teníamos en mente. Reynolds ha estado en el negocio del procesamiento de datos por tres décadas y es bueno en eso, pero nunca debería haber sido gerente de este proyecto".

Conforme a registros obtenidos por el SENTINEL – OBSERVER, Silicon Techtronics, Michael Waterson. Reynolds reemplazaba a John Cramer, quien gerenciaba al anterior proyecto Robbie CX10 y CX20. Cramer fue puesto a cargo del proyecto CX30, pero murió inesperadamente en un accidente aéreo. Al colocar a Reynolds a cargo del proyecto CX30, nos dice nuestra fuente, que Waterson iba en contra del consejo de Ray Jonson, jefe de la División de robótica. De acuerdo con estas fuentes, Jonson se oponía fuertemente a la alternativa de ponerlo a Reynolds como jefe del proyecto Robbie CX30. Estas fuentes dijeron al SENTINEL – OBSERVER que la elección de Waterson por Reynolds fue puramente una decisión de recorte de gastos. Era más barato transferir a Reynolds a la División robótica que incorporar a un nuevo líder de proyecto de fuera de la corporación.

La fuente anónima que el SENTINEL – OBSERVER llamará "Marta" describió la situación de este modo: "Waterson pensaba que sería mas barato transferir a Reynolds a robótica antes que intentar encontrar afuera un nuevo gerente para el proyecto Robbie. Además, Waterson tendía a sospechar de la gente de afuera del grupo. Con frecuencia mandaba memos sobre cuánto tarda la gente en aprender "el modo de hacer las cosas de Silicon Techtronics". Desde el punto de vista de Waterson, Reynolds era el gerente y fue transferido a su nuevo puesto de robótica como un gerente y no como un experto técnico. Claramente, Reynolds se veía a sí mismo tanto gerente como experto técnico. Reynolds no tenia conciencia de sus propias limitaciones técnicas".

Según Marta, Reynolds era muy renuente a gerenciar proyecto que no usaran el modelo de cascada que tan bien le había servido en el procesamiento de datos. Tildó al

modelo prototipo como un "modelo de moda" en la reunión del 11 de diciembre, y después de una serie de intercambios verbales la cosa se puso muy personal.

"Anderson estaba especialmente expresiva", recuerda Marta. "Tenía mucha experiencia con interfaces con usuarios y desde su perspectiva, la interfaz robot – operador era crítica para el éxito del CX30, dado que la intervención del operador seria frecuente y a veces crítica". En su entrevista con el SENTINEL – OBSERVER, Jan Anderson comentó sobre el aspecto de la reunión del 11 de diciembre:

"Reynolds estaba en contra de "perder el tiempo" – para usar sus propias palabras – con cualquier tipo de análisis formal de las propiedades de los factores humanos y su interfaz con el usuario. Para él, la interfaz con el usuario real eran un tema periférico".

"Para él (Reynolds), cualquier cosa nueva era "moda", agrega Anderson. "Las interfaces de las computadoras eran una moda, el diseño orientado a objetos era una moda, la especificación formal y las técnicas de verificación eran una moda, y por sobre todo, el modelo en prototipo era una moda". Justo una semana después de la reunión del 11 de diciembre, el grupo de proyecto Robbie recibió un memo de Sam Reynolds concerniente al plan para el proyecto Robbie CX30.

"Era el modelo de cascada, como salido de un libro", Anderson dijo a este reportero mientras revisaba una copia del memo con el plan del proyecto. "Análisis de requerimiento y especificación, luego diseño de arquitectura y diseño detallado, codificación, prueba, entrega y mantenimiento. En el modo de ver de Reynolds, no hacía falta tener ninguna interacción del usuario con el sistema hasta muy, pero muy avanzado el proyecto". El SENTINEL – OBSERVER se ha enterado de que el primer operador que realmente uso a Robbie CX30 en una función industrial fue Bart Matthews, el hombre que fue muerto en la tragedia del robot asesino. Este primer uso de Robbie CX30 en un uso industrial fue cubierto por los medios, incluyendo este periódico. Como una gran ironía, el Informe Anual de Silicon Techtronics para los accionistas, publicado el pasado mes de marzo, contiene en la brillante portada una foto de un sonriente Bart Matthews. A Matthews se lo muestra operando al mismísimo robot Robbie CX30 que aplastó hasta tan solo dos meses después de la toma fotográfica.

7.3.5. Articulo 5 - Silicon Techtronics prometió entregar un robot seguro. Cuestionada la calidad del entrenamiento del operador.

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

En una conferencia de prensa de esta tarde, un grupo de programadores que se autodenominan "Comité de Justicia para Randy Samuels", distribuyo documentos que

muestran que Silicon Techtronics se obligo a sí misma a hacer entrega de robots que "no causarían daño corporal a los operadores humanos". Randy Samuels es el programador que ha sido acusado del asesinato infame del caso del "robot asesino".

"No podemos entender como el Fiscal pudo acusar a Randy con esos cargos cuando, de hecho, la compañía Silicon Techtronics estaba legalmente obligada a producir y entregar robots seguros a Cybernetics", dijo el vocero del comité, Ruth Witherspoon. "Creemos que en todo esto hay un encubrimiento y que hay algún tipo de confabulación entre la gerencia de SiliTech (Silicon Techtronic) y la oficina del Fiscal. Michael Waterson era uno de los mas grandes contribuyentes de la campaña de reelección de la Sra. McMurdock del año pasado". Michael Waterson es presidente ejecutivo de Silicon Techtronics. Jane McMurdock es la Fiscal de la ciudad de Silicon Valley. El SENTINEL – OBSERVER confirmó que Waterson hizo varios grandes aportes a la campaña de reelección de McMurdock del otoño pasado.

"A Randy le están haciendo pagar los platos rotos por una empresa que tiene estándares de control de calidad laxos y no lo vamos a permitir! Grito Whiterspoon en una declaración a los periodistas. "Creemos que la política ha entrado en todo esto".

Los documentos que fueron distribuidos por el comité por la "Justicia para Randy Samuels" eran porciones de lo que se llama un "documento de requerimientos". Según Ruth Whiterspoon y otros miembros del comité, este documento prueba que Samuels no fue legalmente responsable de la muerte de Bart Matthews, el desafortunado operador de robot que fue muerto por un robot de Silicon Techtronics en Cybernetics, Inc. en Silicon Heights el pasado mes de abril.

El documento de requerimientos es un contrato entre Silicon Techtronics y Cybernetics, Inc. específica con total detalle la funcionalidad del robot Robbie CX30 que Silicon Techtronics prometió entregar a Cybernetics. Según Whiterspoon, el robot Robbie CX30 fue diseñado para ser un robot "inteligente" que pudiera ser capaz de operarse en una variedad de funciones industriales. Cada cliente de la corporación necesitó de los requerimientos separados ya que Robbie CX30 no era un "robot de llave en mano" sino un robot que necesitaba ser programado de forma diferente para cada aplicación.

No obstante, todos los documentos de requerimientos que fueron acordados bajo los auspicios del proyecto Robbie CX30, incluyendo al acuerdo entre Silicon Techtronics y Cybernetics, contiene los siguientes fundamentos de importancia:

- "El robot será de operación segura y aún bajo circunstancias excepcionales (ver Sección 5.2), el robot no causará daño corporal alguno al operador humano".

- "En el caso de condiciones excepcionales que potencialmente contengan el riesgo de daño corporal (ver Sección 5.2.4 y todas las subsecciones), el operador humano podrá ingresar una secuencia de códigos de comando, según se describe en las secciones relevantes de las especificaciones funcionales (ver Sección 3.5.2.), que detendrá el movimiento del robot mucho antes que pueda ocurrir un daño corporal".
- Las "condiciones excepcionales" incluyen eventos inusuales tales como datos extraños desde los sensores del robot, movimiento errático o violento del robot o error del operador. Fue justamente esa condición excepcional la que llevó a la muerte a Bart Matthews.

Estos párrafos fueron extractados de las porciones del documento de requerimientos que trata sobre los "requerimientos no funcionales". Los requerimientos no funcionales listan en detalle las restricciones bajo las cuales operaría el robot. Por ejemplo, el requerimiento de que el robot seria incapaz de dañar a su operador humano es una restricción y Silicon Techtronics, según Ruth Whiterspoon, estaba legalmente obligada a satisfacer este punto. La parte de los requerimientos funcionales del documento de requerimientos cubre (nuevamente en sumo detalle) el comportamiento del robot y su interacción con el entorno y su operador humano. En particular, los requerimientos funcionales especificaban el comportamiento del robot bajo cada una de las condiciones excepcionales esperadas. En su declaración a los periodistas en la conferencia de prensa, Whiterspoon explicó que Bart Matthews fue muerto cuando se produjo la condición excepcional 5.2.4.26. Esta involucra un movimiento del brazo del robot extremadamente violento e impredecible. Esta condición requiere de la intervención del operador, a saber el ingreso de los códigos de comandos mencionados en los documentos, pero aparentemente Bart Matthews se confundió y no pudo ingresar con éxitos los códigos.

"Si bien el programa de Randy Samuels estaba mal – él en verdad malinterpreto las formulas de la dinámica del robot, como se informó en los medios. La condición excepcional 5.2.4.26 estaba diseñada para proteger justamente este tipo de contingencia", dijo Whiterspoon a los periodistas. "Los valores del movimiento del robot generados por el programa de Randy identificaron correctamente a esta condición excepcional y el operador del robot recibió el debido aviso de que algo andaba mal".

Whiterspoon dijo que poseía una declaración formada de otro operador de robot de Cybernetics en donde afirmaba que las sesiones de entrenamientos ofrecidas por Silicon Techtronics nunca mencionaron a ésta ni a tantas otras condiciones excepcionales. Según Whiterspoon, el operador del robot ha jurado que ni a él ni a ningún otro operador de robots les fue dicho que jamás el brazo del robot podía oscilar violentamente.

Whiterspoon citó esta declaración en la conferencia de prensa. "Ni yo ni Bart recibimos entrenamiento para manejar este tipo de condición excepcional. Dudo mucho que Bart Matthews tuviese idea de que se suponía que debía hacer cuando la pantalla de la computadora comenzó a mostrar mensaje de error".

Las condiciones excepcionales que requieren de la intervención del operador causan un mensaje de error que se genera en la consola del operador. La Policía de Silicon Valley afirmó que cuando Bart Matthews fue muerto, el manual de referencia en su consola estaba abierto en la página del índice que contenía los códigos de ingreso para "errores".

Whiterspoon luego citó, secciones del documento de requerimientos que obligan a Silicon Techtronics (el vendedor) a entrenar adecuadamente a sus operadores:

- "El vendedor suministrará cuarenta (40) horas de entrenamiento para los operadores. Este entrenamiento cubrirá todos los aspectos de la operación del robot, incluyendo una cobertura exhaustiva que contenga potencialmente el riesgo de daño corporal. El vendedor proveerá y administrará instrumentos de prueba apropiados que serán usados para certificar el suficiente entendimiento del operador de las operaciones de la consola del robot y de los procedimientos de seguridad. Solo los empleados del cliente que hayan pasado estas pruebas estarán habilitados para operar el robot Robbie CX30 en una verdadera función industrial.
- "El manual de referencia deberá suministrar instrucciones claras para la intervención del operador en todas las situaciones excepcionales, especialmente e inclusive aquellas que contengan potencialmente el riesgo de daño corporal".

Según Whiterspoon, las declaraciones juradas de varios operadores de robots de Cybernetics Inc., aseguran que solo se destino un día laborable (aproximadamente 8 horas) al entrenamiento de operadores. Es mas, casi no dedicó tiempo alguno al tratamiento de condiciones excepcionalmente peligrosas.

"La prueba escrita desarrollada por Silicon Techtronics para habilitar a un operador están considerada por los empleados de Cybernetics como un chiste", aseguro Whiterspoon, "Obviamente Silicon Techtronics no le dedicó mucho tiempo al entrenamiento y procedimientos de pruebas obligatorios según el documento de requerimientos según la evidencia en nuestro poder".
Reimpreso con el permiso de ROBOTIC WORD el diario de ROBOTICS AND ROBOTICS APPLICATIONS.

7.3.6. Articulo 6 - La interfaz del "Robot Asesino"

Dr. Horace Gritty, Departamento de Ciencias de la Computación y materias relacionadas. Universidad de Silicon Valley, Silicon Valley, EEUU

Resumen: El robot industrial Robbie CX30 se suponía que debía restablecer un nuevo modelo de inteligencia de robots industriales. Desgraciadamente, uno de los primeros robots Robbie CX30 mató a un obrero de la línea de montajes, y esto llevó a la acusación de uno de los que desarrollaron el robot, Randy Samuels. Este paper promueve la teoría de que quien debería estar en juicio es el desarrollador de la interfaz robot – operador. El robot Robbie CX30 viola casi todas las reglas del diseño de interfaz. Este paper se centra en como la interfaz del Robbie CX30 violó cada una de las "Ocho reglas de oro" de Shneiderman.

1. Introducción

El 17 de mayo de 1992, un robot industrial Robbie CX30 de Silicon Techtronics mató a su operador, Bart Matthews, en Cybernetics Inc., en Silicon Heights, un suburbio de Silicon Valley. La investigación de los hechos del accidente guiaron a las autoridades a la conclusión de que un módulo de software, escrito y desarrollado por Randy Samuels, un programador de Silicon Techtronics, fue el responsable del comportamiento errático y violento que a su vez llevo a la muerte por decapitación a Bart Matthews

NOTA: Los medios de prensa manejaron la información haciendo creer que Bart Matthews había sido aplastado por el robot, pero la evidencia fotográfica dada a este autor muestra otra cosa. Tal vez, las autoridades trataban de proteger la sensibilidad pública.

Como experto en el área de interfases con el usuario (1, 2, 3), se me pidió prestar ayuda a la policía en la reconstrucción del accidente. Para poder hacer esto, se le pidió a Silicon Techtronics que me suministrara un simulador de Robbie CX30 que incluyera por completo la consola del operador del robot. Esto me permitió investigar el comportamiento del robot sin tener que en realidad arriesgarme seriamente. Debido a mi profundo entendimiento de interfaces con el usuario y factores humanos pude reconstruir el accidente con extrema precisión. Sobre la base de esta reconstrucción, llegué a la conclusión de que fue el diseño de la interfaz y no el imperfecto diseño del software lo que debería haber sido visto como el criminal en este caso.
A pesar de mi descubrimiento, la fiscal Jane McMurdock insistió en proseguir el caso en contra Randy Samuels. Pienso que cualquier computador científico competente, dado una oportunidad de interactuar con el simulador del Robbie CX30, también habría concluido que el diseñador de la interfaz y no el programador debería haber sido acusado por negligencia, si no por homicidio no premeditado.

2. "Las ocho reglas de oro" de Schneiderman

Mi evaluación de la interfaz con el usuario del Robbie CX30 está basada en las "ochos reglas de oro" de Scheiderman (4), también utilicé otras técnicas para evaluar la interfaz, pero estas serán publicadas en papers separados. En esta sección ofrezco una breve revisión de las ocho reglas de oro de Scheiderman, un tema que resultará más familiar para los expertos en interfaces de computación como yo y no hackers de robótica que leyeron este oscuro periódico.

Las ocho reglas de oro son:

1. Buscar siempre la coherencia. Tal como se puede ver mas abajo, es importante que una interfaz con el usuario sea coherente a muchos niveles. Por ejemplo, los diseños de pantalla deben ser coherentes de una pantalla a la siguiente. En un entorno en que se usa una interfaz grafica con el usuario (GUI), esto también implicará concordancia de un utilitario al siguiente.
2. Permitirle a los usuarios frecuentes el uso de shortcuts. Los usuarios frecuentes (o,´power users´) pueden desalentarse ante tediosos procedimientos. Permitirles a estos usuarios un procedimiento menos tedioso para completar una tarea dada.
3. Dar realimentación de información. Los usuarios necesitan ver las consecuencias de sus acciones. Si un usuario ingresa un comando pero la computadora no muestra yá lo que sea que esté procesando o lo que ha procesado, esto puede dejar al usuario confundido o desorientado.
4. Diseñar diálogos que tengan un fin. Interactuar con una computadora es algo así como dialogar o conversar. Cada tarea debe tener un inicio, un desarrollo y un fin. Es importante que el usuario sepa cuando una tarea esta finalizada. El usuario necesita tener la sensación de que la tarea ha alcanzado su término.
5. Permitir manejos simples de errores. Los errores del usuario deben estar diseñados dentro del sistema. Otro modo de decirlo es que no debe haber ninguna acción por parte del usuario que sea considerada como un error que está mas allá de la capacidad del sistema para manejarlo. Si el usuario comete un error, debe recibir información útil, clara y concisa sobre la naturaleza de tal error. Debe resultar fácil para el usuario deshacer este error.
6. Permitir deshacer las acciones con facilidad. Mas genéricamente, los usuarios deben poder deshacer lo que han hecho, sea esto o no de la naturaleza errónea.
7. Respaldar que el centro del control este internamente. La satisfacción del usuario es alta cuando el usuario tiene la sensación de que es él o ella quien tiene el control y la satisfacción del usuario es baja cuando el usuario siente que la computadora tiene el control. Diseñar interfaces para reforzar la sensación de que es en el usuario donde yace el control del ámbito de la interacción humano – computadora.

8. Reducir la carga de la memoria inmediata. La memoria inmediata del ser humano es notablemente limitada. Los psicólogos regularmente citan la ley de Miller que dice que la memoria inmediata está limitada a siete piezas discretas de información. Hacer todo lo posible para liberar la carga en la memoria del usuario. Por ejemplo, en lugar de pedirle al usuario que tipee el nombre de un archivo para abrirlo, presentar al usuario una lista de los archivos disponibles en ese momento.

3. Revisión de la consola del robot

La interfaz del operador Robbie CX30 violó casi todas y cada una de las reglas de Shneiderman. Muchas de estas violaciones fueron directamente responsables del accidente que terminó con la muerte del operador del robot.

La consola del robot era una IBM PS/2 modelo 55SX con un procesador 80386 y un monitor EGA color con una resolución de 640 X 480. La consola tenía un teclado, pero no mouse. La consola está empotrada en una estación de trabajo que tenía, además, estantes para manuales y un área para tomar notas y para leer manuales. No obstante, el área para escribir/ leer estaba a bastante distancia de la pantalla de la computadora, o sea que era bastante incómodo y cansador para el operador manejar cualquier tarea que requiera de mirar algo en el manual y luego actuar rápidamente con respecto a la posición de la consola en el área de escribir/ leer. Esto resentía mucho la espalda del operador y también causaba excesivos cansancios a la vista.

No puedo comprender como un sistema sofisticado como éste no pudo incluir un aparato de mejor diseño para los ingresos de datos. Uno solo podría concluir que Silicon Techtronics no tenía mucha experiencia en tecnología de interfaces con el usuario. El documento de requerimientos (5) especificaba un sistema manejado por menús, lo cual era una elección razonable. Sin embargo, en un utilitario en donde lo esencial era la rapidez, especialmente cuando la seguridad del operador esta en juego, el uso de un teclado para todas las tareas de selección de opción de menús fue una elección de extremada pobreza, que requería de mucho uso del teclado para lograr el mismo efecto que podía haberse conseguido casi instantáneamente mediante un mouse. (Ver el paper escrito por Foley et al. (6) En realidad, se me ocurrieron todas estas ideas antes que Foley las publicara, pero él me ganó).

El operador del robot podía interactuar con el robot y de este modo producir un impacto sobre su comportamiento al hacer las opciones en un sistema de menús. El menú principal consistía de 20 ítems, demasiados en mi opinión, y cada ítem del menú principal tenia un submenú tipo desplegable asociado a este. Algunos de los submenús tenían tanto como veinte ítems – nuevamente demasiados. Es más, parecía haber poca lógica en cuanto a por qué los ítems de los menús estaban listados en el orden en que estaban. Hubiese sido mucho mejor una organización alfabética o funcional.

Alguno de los ítems en los menús desplegables tenían hasta cuatro menús pop up relacionados a estos. Estos aparecían en secuencias a medida que se hacía una selección correspondiente en los submenús. Ocasionalmente, una elección de un submenú abriría un cuadro de dialogo en la pantalla. Un cuadro de dialogo requiere de cierto tipo de interacción entre el operador y el sistema, por ejemplo resolver ciertos temas, como ser, ingresar cuál es el diámetro de un dispositivo dado a ser bajado en el baño de ácido.

El sistema de menús presenta una estricta jerarquía de elección de menús. El operador podría volver hacia atrás en esta jerarquía apretando la tecla de escape. Esta tecla escape también podría terminar los diálogos.
El uso del color en la interfaz fue muy poco profesional, había demasiados colores en un espacio demasiado chico. Los contrastes eran muy fuertes y el resultado, para este revisor, resultó en una severa fatiga ocular en tan solo quince minutos de usos. Hubo uso excesivo de efectos musicales tontos y falsees cuando se ingresaba opciones o códigos erróneos.

Uno debería preguntarse por qué Silicon Techtronics no intentó un enfoque mas sofisticado para el diseño de interfaz. Luego de un cuidadoso estudio del dominio de los utilitarios del Robbie CX30, he llegado a la conclusión de que una interfaz de manipulación directa, que mostrara literalmente al robot en la consola del operador, habría sido lo ideal. El entorno tan visual dentro del cual operaba el robot se prestaba naturalmente al diseño de metáforas de pantallas apropiadas para ese entorno, metáforas que el operador podría entender con facilidad. Esto permitiría que el operador manipulara el robot mediante el manejo, en la consola del robot, de la representación grafica del robot en su entorno. He solicitado a unos de mis estudiantes en el doctorado, Susan Farnsworth, que investigará un poco más esta posibilidad.

4. En qué modo la interfaz del robot Robbie CX30 violó las ocho reglas de oro

La interfaz con el usuario de Robbie CX30 violó todas las reglas de oro en diferentes modos. En este paper sólo trataré unas pocas instancias de violaciones de reglas, dejando la discusión más detallada para futuros artículos y mi próximo libro.

NOTA:. CODEPENDENCIA, Cómo los Usuarios de Computadoras permiten deficientes Interfaces con el Usuario, Angst Press, Nueva York. Este libro presenta una teoría radicalmente nueva con respecto a la relación entre la persona y la máquina. Esencialmente, algunas personas necesitan una interfaz de mala calidad a los fines de evitar ciertos problemas psicológicos no resueltos en sus vidas.

Lo que haré es destacar esas violaciones que fueron relevantes en este accidente en particular:

- Buscar siempre la coherencia: En la interfaz del usuario de Robbie CX30 hubieron muchas incoherencias. Los mensajes de error podían aparecer en casi cualquier color y podían estar acompañados por casi cualquier tipo de efecto musical. Además, los mensajes de error podían aparecer en casi cualquier lugar de la pantalla. Cuando Bart Matthews vio el mensaje de error de la condición excepcional que ocurrió luego, la cual requería la intervención del operador, es probable que fuera esa la primera vez que veía ese mensaje en especial. Además, el mensaje de error apareció en un cuadro verde, sin ningún efecto de sonido. Este es el único mensaje de error de todo el sistema que aparece en verde y sin ningún tipo de acompañamiento de orquesta.
- Permitir que los usuarios frecuentes utilicen shortcuts: Este principio no aparece de ningún modo en todo el diseño de la interfaz. Por ejemplo, hubiera sido una buena idea permitir que los usuarios frecuentes pudieran ingresar la primera letra de la opción de un submenú o menú en lugar de requerírseles el uso de las teclas del cursor y luego la tecla "enter" para elegir esa opción determinada. El mecanismo de selección de menús de este mismo sistema debe haber provocado al operador bastante fatiga mental. Es más, debería haberse permitido algún tipo de sistema de tipeo anticipado que permitiera al usuario frecuente ingresar una secuencia de opciones de menú sin tener que esperar a que apareciera realmente el menú en pantalla.
- Ofrecer realimentación de información: En muchos casos el usuario no tiene idea de si el comando que acaba de ingresar se está procesando o no. Este problema se exagera además por las inconsistencias en el diseño de la interfaz con el usuario. En algunos casos al operador se le da una realimentación detallada de lo que el robot está ejecutando. En otros, el sistema permanece misteriosamente silencioso. En general, al usuario se lo lleva a que espere algún tipo de realimentación y por consiguiente se queda confundido cuando está no se le da. En la pantalla, no hay una representación visual del robot y su entorno, y la visión que tiene el operador del robot a veces está obstruida.
- Diseñar diálogos que tengan fin: Hay muchos casos en los que una secuencia dada de tecleado representa una idea holística, una tarea completa, pero al operador se lo deja sin el tipo de realimentación que le confirmare que la tarea ha sido en efecto completada. Por ejemplo, hay un dialogo bastante complicado que se necesita cuando se quiere sacar un elemento de un baño de ácido. Sin embargo, luego de completar este dialogo, el usuario es llevado a otro dialogo nuevo, y no relacionado con este, sin que se les informe que el dialogo anterior ha finalizado.
- Ofrecer manejo simple de los errores: El sistema pareciera estar diseñado para que el usuario se lamentara por cualquier ingreso erróneo. No sólo el sistema permite numerosas oportunidades para el error, sino que cuando un error en realidad ocurre, es probable que se repita por algún tiempo. Ello se debe a que la

interfaz con el usuario, fue diseñada en forma tal que corregir el error era una odisea tediosa, frustrante y a veces enfurecedora. Algunos de los mensajes de error eran directamente ofensivos y condescendientes.

- Permite deshacer las acciones con facilidad: Como se menciona en el párrafo anterior, la interfaz con el usuario hace muy difícil la tarea de deshacer entradas erróneas. En general, el sistema de menús permite deshacer fácilmente las acciones, pero esta filosofía no alcanza para el diseño de los cuadros de diálogos y al manejo de condiciones excepcionales. Desde un punto de vista práctico (opuesto al teórico), la mayoría de las acciones son irreversibles cuando el sistema está en un estado de condición excepcional, y esto ayudó a llegar a la tragedia del robot asesino.
- Promover que uno sea el centro del robot: Muchas de las deficiencias tratadas en los párrafos precedentes disminuyeron la sensación de "tener el control". Por ejemplo, no recibir información, no poder concluir con las interacciones, no permitir deshacer con facilidad las acciones en el momento que surgen las excepciones, todas estas cosas actúan para disminuir la sensación de que el usuario posee el control sobre el robot. Hubieron muchas características de esta interfaz que hicieron que el operador sintiera que hay un enorme bache entre la consola del operador y el robot en sí, mientras que un buen diseño de interfaz hubiera hecho transparente la interfaz con el usuario y le hubiere dado al operador del robot la sensación de estar en contacto directo con el mismo. En un caso, le ordené al robot mover un elemento desde el baño de ácido hasta la cámara de secado y pasaron 20 segundos antes de que el robot pareció responder. De este modo, no tuve la sensación de estar controlando al robot. Tanto la repuesta demorada del robot como la falta de realimentación en la pantalla de computadora, me hicieron sentir que el robot era un agente autónomo, la verdad un sentimiento como mínimo perturbador.
- Reducir la carga de memoria de corto plazo: Un sistema que se maneja por medio de menús es generalmente bueno en términos de carga de memoria que crea a los usuarios. No obstante, hay gran variación entre implementaciones particulares de sistemas de menú en lo que hace a carga de memorias. La interfaz con el usuario de Robbie CX30 tenía menúes muy grandes sin ninguna organización interna. Esto crea una gran carga al operador en términos de memoria y también en términos de tiempo de búsqueda, el tiempo que lleva al operador ubicar una opción determinada del menú.
- Muchas pantallas de dialogo requerían que el usuario ingresara con el teclado números de partes, nombre de archivos, y otra información. El sistema podría haberse diseñado fácilmente de forma de mostrarle al usuario estos números de partes, etc., sin la necesidad que el usuario recordara estas cosas de su propia memoria. Esto incrementaba la carga sobre la memoria del usuario.
- Para finalizar, y esto es realmente imperdonable, increíble como pueda parecer ino había ninguna instalación de ayuda en línea o sensible al contexto!. Si bien he ido a los cursos de entrenamientos ofrecidos por Silicon Techtronics, muchas veces me

encontré navegando por los manuales de referencia para poder encontrar la respuesta aún a las más básicas preguntas, tales como: "¿Qué significa esta opción de menú? ¿Qué pasa si selecciono esta opción?".

5. Una reconstrucción de "la tragedia del robot asesino".

Las fotos policiales de la escena del accidente no son nada agradables de ver. La consola del operador estaba salpicada con bastante cantidad de sangre. No obstante, la calidad de las fotos es excepcional y utilizando técnicas de ampliación pude descubrir los siguientes factores de importancia sobre el momento en que fue decapitado Bart Matthews:

- La luz NUM LOCK estaba encendida: El teclado de IBM contiene un tablero que se puede operar de dos modos. Cuando la luz de NUM LOCK esta encendida, esa parte se comporta como una calculadora. Del otro modo, las teclas pueden usarse para mover el cursor de la pantalla.
- Había sangre esparcida en el tablero numérico: Las huellas ensangrentadas indican que Bart Matthews estaba usando el tablero numérico en el momento en que fue golpeado y muerto.
- Se encontraba titilando en verde un mensaje de error: Esto nos dice que la situación de error vigente en el momento que ocurrió la tragedia. El mensaje de error decía "ROBOT DYNAMICS INTEGRITY ERROR – 45".
- Había un manual de referencia apoyado y abierto sobre el área de lectura/ escritura de la estación de trabajo.
- Uno de los cuatros volúmenes del manual de referencia estaba abierto en la página del índice que contenía el ítem "ERRORES/ MENSAJES".
- En la pantalla también había un mensaje que mostraba instrucciones al operador
- El mensaje aparecía en amarillo en la parte inferior de la pantalla. En el mensaje se leía: "POR FAVOR INGRESE INMEDIATAMENTE LA SECUENCIA DE COMANDOS PARA CANCELAR EL ERROR DINAMICO DEL ROBOT!!!"

En base a las evidencias físicas mas otras evidencias contenidas en los registros del sistema, y basándose en la naturaleza del error que ocurrió (error de integridad de dinámica del robot – 45, el error que estuvo causado por el programa de Randy Samuels), he llegado a la conclusión de que ocurrió la siguiente secuencia de eventos en la fatal mañana de la tragedia del robot asesino:

- 10:22:30 "ERROR DE INTEGRIDAD DE DINAMICA DEL ROBOT –45" aparece en la pantalla, Bart Matthews no lo ve porque no hay efecto de audio o señal sonora tal como ocurre en todas las situaciones de error. Además, el mensaje de error aparece en verde, lo que en todos los contextos significa que hay proceso completándose con normalidad.
- 10:24:00 El robot comienza a moverse lo suficientemente violento como para que Bart Samuels lo note.

- 10:24:05 Bart Matthews se da cuenta del mensaje de error, no sabe lo que significa. No sabe que hacer. Intenta con el submenú "cancelación de emergencia", un submenú de uso genérico para apagar el robot. Este involucra SEIS opciones de menú por separado, pero el Sr. Matthews no se da cuenta de que la luz del NUM LOCK esta encendida. Por ende, las opciones del menú no estaban ingresando, dado que las teclas del cursor operaban como teclas de calculadora.
- 10:24:45 El robot gira el baño de ácido y comienza a arrastrar la consola del operador, con sus brazos dentados batiéndose con gran amplitud. Nadie espera que un operador tuviera que huir de un robot descontrolado, así que Bart Matthews queda atrapado en su área de trabajo por el robot que avanza. Mas o menos para este momento es que Bart Matthews saca el manual de referencia y empieza a buscar el error ERROR DE INTEGRIDAD DE DINAMINA DEL ROBOT –45 en el índice. Ubica con éxito la referencia a mensajes de error en el índice.
- 10:25:00 El robot ingresa al área del operador. Bart Matthews abandona la búsqueda del procedimiento del operador ante un error de integridad dinámica del robot. En su lugar, intenta una vez mas ingresar la secuencia de "cancelación de emergencia" desde el teclado numérico, momento en que es golpeado.

6. Resumen y conclusiones

Si bien el módulo de software escrito por Randy Samuels causó en verdad que el robot Robbie CX30 oscilara fuera de control y atacara a su operador humano, un buen diseño de la interfaz hubiera permitido al operador terminar con el comportamiento errático del robot. En base al análisis de la interfaz del usuario del robot llevado a cabo utilizando las ochos reglas de oro de Schneiderman, el experto en diseño de interfaces ha arribado a la conclusión de que el diseñador de la interfaz y no el programador fue la parte mas culpable en este desafortunado evento.

7. Referencias

- Gritty, Horace (1990). The only user interface book you´ll ever need. Vanity Press, Oshkosk, WI 212 pag. [El único libro sobre interfaz de usuarios que Usted necesitara].
- Gritty, Horace (1992). What we cant learn from the killer robot [Lo que podemos aprender de un robot asesino], charla dada en el Simposio internacional de la Universidad de Silicon Valley sobre Seguridad en robot e Interfaces de usuario, Marzo de 1991. también por aparecer en las Notas de los alumnos de la Universidad de Silicon Valley].

- Gritty, Horace (se espera para 1993). CODEPENDENDY: How computer users enable poor user interfaces, Angst press, New York [Como los usuarios de computadoras permiten interfaces deficientes].
- Shneiderman, Ben (1987), Designing the user interface, Addison Wesley, reading MA, 448 pag [Diseño de Interfaces].
- DOCUMENTO DE REQUERIMIENTOS DEL ROBOT INDUSTRIAL INTELIGENTE Robbie CX30: versión de Cybernetics INCS., Documento Técnico N° 91-0023XA, Silicon Techtronics Corporation Silicon Valley, USA 1245 pag.
- Foley, J. P., Wallace, V. L., y Chan, P. (1984): The human factors of computer graphics interaction techniques [Los factores humanos de las técnicas de interacción de graficas de computación] IEEE COMPUTER GRAPHICS AND APPLICATIONS, 4(11) pag, 13-48.

7.3.7. Artículo 7 - Ingeniero de software cuestiona la autenticidad de las pruebas de software del "Robot Asesino". La indagación de un profesor de la Universidad de Silicon Valley provoca serios cuestionamientos legales y éticos.

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

El caso del "robot asesino dio un giro significativo ayer cuando un profesor de la Universidad de Silicon Valley presentó un informe que cuestiona la autenticidad de las pruebas que fueron hechas por Silicon Techtronics al software del "robot asesino". El profesor Wesley Silber, profesor de Ingeniería del Software, dijo en una conferencia de prensa realizada en la universidad que los resultados de las pruebas reflejados en los documentos internos de Silicon Techtronics no concordaban con los resultados de las pruebas obtenidos cuando él y sus colegas ensayaron el software real del robot.

Silicon Valley aún está reaccionando por el anuncio del Profesor Silber, que podría jugar un papel importante en el juicio a Randy Samuels, el programador de Silicon Techtronics que fue acusado por homicidio no premeditado en el ahora infame incidente del "robot asesino". Presionada por su reacción por el informe del profesor Silber, la fiscal Jane McMurdock reiteró su confianza en que el jurado encontrara culpable a Ray Samuels. Sin embargo, la Fiscal Jane McMurdock impresionó a los periodistas cuando agregó "pero, esto en verdad promueve la posibilidad de nuevas acusaciones".

Ruth Whirterspoon, la vocero del "Comité de justicia para Randy Samuels", también estuvo exultante cuando habló a este periódico. "McMurdock no puede tener ambas cosas". O el programador es el responsable por esta tragedia o se deberá hacer responsable a la gerencia por ello. Creemos que el informe del Silber exonera a nuestro amigo y colega Randy Samuels".

El gerente Ejecutivo de Silicon Techtronics Michael Waterson hizo la siguiente tibia declaración sobre el informe de Silber:

- "Tan pronto se anunció la acusación de Randy Samuels personalmente le pedí a un estimado ingeniero del software, el Dr. Wesley Silber, que llevara a cabo una indagación objetiva sobre los procedimientos de aseguramiento de la calidad en Silicon Techtronics. Como gerente ejecutivo de este proyecto, siempre he insistido en que la calidad es lo primero, a pesar de lo que hayan podido leer en los periódicos".

- "Le pedí al profesor Silber que condujera una investigación objetiva de todos los aspectos de aseguramiento de la calidad de Silicon Techtronics. Prometí al profesor Silber que tendría acceso a toda la información relevante a esta infortunada situación. Le dije en una reunión frente a frente, en mi oficina, que debía proseguir la investigación hasta su final sin importar a donde terminara, sin importar las implicancias".

- "Basándome en la información que yo recibía de mis gerentes, nunca se me hubiera ocurrido que pudiesen existir problemas de que los procedimientos de aseguramiento de la calidad fueran, ya sea débiles, o estuviesen alterados. Quiero asegurarle al publico que la o las personas responsables de esta falta de aseguramiento de la calidad del software dentro de la División de Robótica de Silicon Techtronics serán exhortados a encontrar trabajo en otro lado".

Roberta Matthews, viuda de Bart Matthews, el operador del robot que fue muerto en el incidente, habló telefónicamente desde su casa con el SENTINEL – OBSERVER. "Aún quiero ver al Sr. Samuels condenado por lo que le hizo a mi marido. No entiendo de dónde viene toda la conmoción. EL hombre que asesinó a mi esposo, debería haber probado su propio software!".

El SENTINEL – OBSERVER entrevistó al profesor Silber justo antes de su conferencia de prensa. En las paredes de su oficina estaban colgados numerosos premios recibidos a raíz de su trabajo en el campo de Ingeniería del Software y aseguramiento de la calidad del software. Comenzamos la entrevista pidiendo al profesor Silber que explicara por qué a veces el software no es confiable. Contestó a nuestra pregunta citando la enorme complejidad del software.

"Los grandes programas de computadora son indiscutiblemente los artefactos más complejos creados por la mente humana", explico el profesor Silber sentado frente a un monitor de grandes dimensiones. "En algún momento un programa de computación está en uno de los tantos estados posibles, y hay imposibilidad práctica de asegurar que el programa se comportará como corresponde en cada uno de esos estados. No tenemos el tiempo suficiente para hacer tal tipo de prueba exhaustiva. De modo que

usamos estrategias de prueba o heurísticas que muy probablemente encontrarán los errores o bugs, si es que existe alguno".

El profesor Silber ha publicado numerosos papers sobre Ingeniería del Software. Estuvo en la primera plana cuando el año pasado publicó su lista de "Aerolíneas a evitar si su vida dependiera de ello". En esa lista se enumeraban las aerolíneas de cabotaje que él consideraba irresponsables por su compra de aviones que están controladas casi por completo por software de computación.

Poco tiempo después de los cargos contra Randy Samuels en el caso del "robot asesino", el gerente ejecutivo de Silicon Techtronics, Michael Waterson, pidió al profesor Silber que condujera una revisión objetiva de los procedimientos de aseguramiento de la calidad de Silicon Techtronics. La intención de Waterson era contrarrestar la mala publicidad de su empresa luego de las acusaciones de Samuels.

"El aseguramiento de la calidad" se refiere a aquellos métodos que usa un especialista de desarrollo de software para asegurar que el software es confiable, correcto y robusto. Estos métodos se aplican a todo lo largo del ciclo de vida de desarrollo del producto de software. En cada etapa se aplican los métodos de aseguramiento de calidad adecuados. Por ejemplo, cuando un programador escribe código, una medida de aseguramiento de la calidad es probar el código confrontándolo en verdad con los datos de prueba. Otro método sería correr programas especiales, llamados analizadores estáticos, confrontándolos con el nuevo código. Un analizador estático es un programa que busca patrones sospechosos en los programas, patrones que podrían indicar errores o bug.
Estas dos formas de aseguramiento de la calidad son denominadas pruebas dinámicas y pruebas estáticas, respectivamente.

El software consiste de componentes discretos o unidades que eventualmente se combinan para crear un sistema mas grande. Las unidades mismas deben ser probadas, y este proceso de prueba individual de las unidades es llamado prueba unitaria. Cuando las unidades se combinan, se deben probar los subsistemas integrados y este proceso se llama prueba de integración.

El profesor Silber comentó al SENTINEL – OBSERVER sobre su trabajo en Silicon Techtronics: "Mike (Waterson) me dijo de ir allí (a la compañía) y conducir una revisión de procedimientos de pruebas de software y de hacer públicos mis hallazgos. Mike parecía confiado, tal vez debido a lo que le habían dicho sus gerentes, en el sentido de que no encontraría nada malo en los procedimientos de aseguramiento de calidad de Silicon Techtronics".

Luego de arribar a Silicon Techtronics, el profesor Silber centró su atención en los procedimientos para ensayo dinámico de software en la compañía.

Ayudado por un grupo de graduados, el profesor Silber describió una discrepancia entre el comportamiento real de la sección del código del programa (escrito por Randy Samuels) que causó que el robot Robbie CX30 matara a su operador, y el comportamiento según se lo registro en la documentación de pruebas de Silicon Techtronics. Este descubrimiento en realidad fue hecho por Sandra Henderson, una estudiante graduada en Ingeniería del Software que está contemplando su doctorado con el profesor Silber. Entrevistamos a la Sra. Henderson en uno de sus laboratorios de computación para egresados en la Universidad de Silicon Valley. "Encontramos un problema en la prueba de unidad", explicó la Sra. Henderson, "Acá están los resultados de la prueba que nos dio el Sr. Waterson en Silicon Techtronics, que se suponen están hechos para código C (lenguaje de programación) que Randy Samuels escribió y que causó el incidente del robot asesino. Como puede ver, todo está claramente documentado y organizado. Hay dos juegos de pruebas: Uno basado en una prueba de caja blanca y otro en una prueba de caja negra. Basándonos en nuestros propios estándares para probar software, estos juegos de prueba están bien diseñados, completos y rigurosos. "La prueba de caja negra implica ver la unidad de software (o sus componentes) como una caja negra que tiene comportamientos predecibles de input y output. Si en el juego de pruebas el componente demuestra los comportamientos esperados para los inputs, entonces pasa la prueba. Los juegos de prueba están diseñados para cubrir todos los comportamientos "interesantes" que una unidad podría mostrar pero sin tener conocimiento alguno sobre la estructura o naturaleza del código en realidad. La prueba de caja blanca implica cubrir todos los pasos posibles a través de la unidad. Así, la prueba de caja blanca se hace con vasto conocimiento de la estructura de la unidad. En la prueba de caja blanca, el juego debe causar que cada sentencia del programa se ejecute por lo menos una vez de modo que ninguna quede sin ser ejecutada.

Sandra Henderson prosiguió explicando el significado de la prueba del software. " Ni la prueba de caja blanca ni de caja negra "prueban" que un programa esté correcto. No obstante, los probadores de software, tales como se emplean en Silicon Techtronics, pueden volverse bastantes expertos en el diseño de los casos de prueba para descubrir nuevos bugs en el software. La actitud apropiada es que una prueba es exitosa cuando se encuentra un bug". Básicamente, al probador le dan un juego de especificaciones y hace lo mejor de sí para demostrar que el código que esta probando no satisface sus especificaciones", explicó la Sra. Henderson.

La Sra. Henderson luego mostró a este reportero los resultados de las prueba que ella en verdad obtuvo cuando corrió el código critico del "robot asesino" usando los juegos de prueba de la compañía, tanto para ensayo de caja blanca como de caja negra. En muchos casos, los resultados registrados en los documentos de prueba de la compañía no fueron los mismos que los generados por el verdadero código del robot asesino.

Durante su entrevista de ayer con el SENTINEL – OBSERVER, el profesor Silber discutió la discrepancia. "Verá, el software que en verdad fue entregado junto con el robot Robbie CX30 no fue el mismo que supuestamente fue probado, ¡por lo menos de acuerdo con estos documentos!. Hemos podido determinar que el verdadero "código asesino", tal como lo llamamos, fue escrito después de que se condujeron supuestamente las pruebas de software. Esto sugiere varias posibilidades. Primero, el proceso de prueba del software, por lo menos para esta parte critica del software, fue falseado deliberadamente. Todos sabemos que hubo una enorme presión para tener listo a este robot en una fecha determinada. Otra posibilidad es que hubo una cierta dificultad en la versión de la gerencia en Silicon Techtronics, en cuanto a que el código correcto fue verdaderamente escrito, y probado con éxito, pero en el producto entregado se insertó el código equivocado".

Solicitamos al profesor Silber que explicara qué quería decir con "versión de la gerencia". "En un proyecto dado, un componente dado de software puede tener varias versiones, versión 1, versión 2, etc.".

Esto refleja la evolución de ese componente a medida que avanza el proyecto. Se necesita tener algún tipo de mecanismo para tener control de las versiones de los componentes de software en un proyecto tan complejo como este. Tal vez el probador de software probó una versión correcta del código de dinámica del robot, pero en realidad se entregó una versión equivocada del mismo. No obstante, esto trae a colación una pregunta en cuanto a qué pasó con el código correcto."

El profesor Silber se reclinó en su sillón. "Realmente esto es una gran tragedia. Si el código asesino hubiese sido pasado por el proceso de prueba de modo honesto, el robot nunca hubiese asesinado a Bart Matthews. Entonces, la pregunta es, ¿qué pasaba en Silicon Techtronics que no permitió una prueba honesta del código critico?"

El SENTINEL – OBSERVER preguntó al profesor Silber si estaba de acuerdo con el concepto de que la interfaz del usuario fue la primordial culpable en este caso. "No creo en el argumento que esgrime mi colega, el profesor Gritty, que toda la culpabilidad en este caso pertenece al diseñador o diseñadores de la interfaz. Concuerdo con ciertas cosas que dice, pero no con todo. Debo preguntarme a mí mismo si Silicon Techtronics estaba poniendo mucho énfasis en la interfaz del usuario como la última línea de defensa contra el desastre. Esto es, ellos sabían que había un problema con la dinámica del robot, pero pensaron que la interfaz podría permitirle al operador manejarlo."

El SENTINEL – OBSERVER preguntó entonces al profesor Silber sobre los cargos que se hacían en cuanto que nunca deberían haber aceptado la designación de Waterson para conducir una investigación objetiva del accidente. Las críticas señalan que la Universidad de Silicon Valley, y en particular el profesor Silber, tenían muchos

intereses comunes con Silicon Techtronics, y de ese modo no podía ser elegido para conducir una investigación objetiva.

"Pienso que mi informe habla por mi mismo, " replicó el profesor Silber, visiblemente molesto por nuestra pregunta. "Ya les he dicho a Ustedes los periodistas una y otra vez que no se trato de una investigación gubernamental sino de una interna de la corporación, de modo que creo que Silicon Techtronics tenia derecho a elegir a quien se le ocurriera, creo que yo les resultaba una persona con integridad."
Ayer tarde, Sam Reynolds, el gerente del proyecto del CX30 contrató una abogada, Valerie Thomas. La Sra. Thomas hizo estas declaraciones a favor de su cliente: " Mi cliente esta escandalizado de que alguien de Silicon Techtronics haya podido engañar al profesor Silber en lo que concierne a las pruebas de software del robot Robbie CX30. El Sr. Reynolds asegura que el software fue probado y que él y otros sabían muy bien el hecho de que había algo que no funcionaba en el software de dinámica del robot. Sin embargo, el Sr. Ray Jonson, el superior inmediato de mi cliente en Silicon Valley, decidió que el robot fuera entregado a Cybernetics, Inc., basándose en la teoría del Sr. Johnson: "Nada es tan blanco como la nieve".

Conforme con esta teoría, el software estaba casi libre de bugs y por ende podía ser liberado. Según el Sr. Johnson, el riesgo de falla era muy pequeño y el costo por demorar más la entrega del robot era muy alto. "Según mi cliente, el Sr. Johnson creyó que las condiciones del medio ambiente que podría llegar a disparar un comportamiento errático y violento del robot eran extremadamente improbables de ocurrir. Aún más, el Sr. Jonson creyó que el operador del robot no podría estar en peligro debido a que la interfaz del usuario fue diseñada de modo de permitir al operador detener el robot fijo en sus guías en el caso de un movimiento del robot que comprometiera la vida del operador."

El Sr. Jonson, jefe de la División de Robótica de Silicon Techtronics, no pudo ser ubicado para obtener sus comentarios. Randy Samuels será juzgado el mes entrante en la Corte de Silicon Valley. Cuando se lo contactó por teléfono, Samuels derivó todas las preguntas a su abogado, Alex Allendale.

Allendale tenía esto para decir con respecto a los descubrimientos del profesor Silber: "Mi cliente remitió el software en cuestión del modo usual junto con la documentación usual y con la esperanza de que su código fuera probado exhaustivamente. Desconocía hasta el momento de que saliera a la luz el informe del profesor Silber, que el código involucrado en esta terrible tragedia no había sido probado adecuadamente o que los resultados de prueba pudieran haber sido falsificados".

"El Sr. Samuels nuevamente quiere expresar su gran pesar por este accidente. Él, más que nadie, quiere que se haga justicia en este caso. El Sr. Samuels nuevamente extiende sus mas sentidas condolencias a la Sra. Matthews y a sus hijos".

7.3.8. Artículo 8 - Empleado de Silicon Techtronics admite falsificación de las pruebas de software. Mensajes tomados del correo electrónico revelan nuevos detalles en el caso del "Robot Asesino". Una asociación de computadores científicos lanza una investigación sobre violaciones al código de ética

Especial para el SENTINEL – OBSERVER de Silicon Vallley. Silicon Valley, EEUU por Mabel Muckraker

Cindy Yardley, una probadora de Silicon Techtronics admitió hoy que ella fue la persona que creó las pruebas de software fraudulentas del "robot asesino". Las pruebas fueron reveladas a principios de la semana por el profesor Wesley Silber de la Universidad de Silicon Valley, con lo que se ha dado en llamar "El informe Silber".

Se cuestionan los procedimientos de aseguramiento de la calidad que fueron realizados en el código del programa escrito por Randy Samuels, el programa acusado por asesinato no premeditado en el incidente del robot asesino. El Informe Silber afirma que los resultados de las pruebas reflejados en documentos internos de Silicon Techtronics son inconsistentes con respecto a los resultados de las pruebas obtenidas cuando fue probado el verdadero código del robot asesino.

Ayer al mediodía, en un acontecer inesperado, anunció su renuncia al cargo de Jefe de Seguridad de Silicon Techtronics, el Sr. Max Worthington, en una conferencia de prensa que fue transmitida en vivo por la CNN y otros informativos.

Worthington sacudió a los periodistas cuando comenzó su conferencia de prensa con el anuncio "Yo soy Marta".

Worthington describió de este modo sus responsabilidades en Silicon Techtronics: "Básicamente, mi trabajo era proteger a Silicon Techtronics de todos los enemigos, locales y extranjeros. Por extranjeros quiero significar adversarios de otras corporaciones. Mi papel era más que nada de dirección. Aquellos que trabajaban bajo mi supervisión tenían muchas responsabilidades, incluyendo la de proteger la planta en si, estar alertas por espionaje industrial e incluso sabotaje. También yo era responsable de vigilar a los empleados que pudiesen estar abusando de drogas o que de algún modo estuviesen siendo desleales con Silicon Techtronics." Luego Worthington apuntó a una pila de volúmenes que había en una mesa a su izquierda. "Estos volúmenes representan tan solo algunos de los relevamientos electrónicos de empleados que yo hice a lo largo de los años para mi superior, el Sr. Waterson. Estas son impresiones de mensajes por e-mail que los empleados de Silicon Techtronics se enviaron entre si y a

personas de otros sitios. Puedo decir con gran certeza que nunca jamás se le dijo a ningún empleado que se hacía este tipo de requisa electrónica. No obstante, creo que la evidencia muestra que algunos empleados sospechaban que esto podía estar pasando."

Varios periodistas preguntaron a los gritos quien en Silicon Techtronics estaba al tanto de esta requisa.

Worthington respondió. "Nadie sabía de esto a excepción del Sr. Waterson y yo, y uno de mis asistentes que era el responsable de conducir el monitoreo. Mi asistente producía un informe especial, resumiendo toda la actividad por e-mail de la semana, y ese informe era para que lo viera Waterson y yo solamente. Si se lo solicitaba, mi asistente podía dar un recuento más detallado de las comunicaciones electrónicas".
Worthington explicó que estaba poniendo a disposición de la prensa las transcripciones del correo electrónico porque quería que saliera a la luz toda la verdad sobre Silicon Techtronics y el incidente del robot asesino.

Los mensajes de e-mail entre empleados de Silicon Techtronics en verdad revelaron nuevas facetas del caso. Un mensaje de Cindy Yardley al Jefe de División de Robótica, Ray Jonson, indica que ella falsificó a su pedido los resultados de las pruebas. Acá esta el texto del mensaje:

A: Ray Jonson
De: Cindy Yardley
Asunto: Software de Samuels
Terminé de crear los resultados de las pruebas de software para ese software problemático, según tu idea de usar una simulación en vez del software propiamente dicho. Adjunto encontrarás el documento de prueba modificado, mostrando la simulación exitosa.
¿Le deberíamos decir a Randy sobre esto?
Cindy

La respuesta de Johnson al mensaje de Yardley sugiere que él sospechaba que el correo electrónico podía no ser seguro.

A: Cindy Yardley
De: Ray Jonson
Asunto: Re: Software de Samuels
Sabía que podía contar contigo. Estoy seguro de que tu dedicación a Silicon Techtronics te será pagada con creces. Por favor, en el futuro usa un medio de comunicación más seguro cuando discutimos este tema. Te aseguro que el modo en que

manejamos esto fue completamente transparente, pero yo tengo mis enemigos acá mismo en la propia SiliTech.
Ray

Estas comunicaciones fueron intercambiadas justo días antes que se enviara al robot Robbie CX30 a Cybernetics Inc.. Este hecho es importante porque las pruebas de software falsificadas no fueron parte de un encubrimiento en el incidente del robot asesino. Estos hechos parecen indicar que el propósito de falsificar las pruebas de software era asegurarse de que el robot Robbie CX30 fuera entregado a Cybernetics, Inc. en la fecha que fue negociada entre Silicon Techtronics y Cybernetics.

Las transcripciones del correo electrónico revelan que hubieron repetidos mensajes de Ray Johnson a diferentes personas en el sentido de que la División de Robótica iba a ser cerrada definitivamente si el proyecto Robbie CX30 no está completado en término. En uno de los mensajes, diserta con su líder de proyecto, Sam Reynolds, acerca de la "teoría Ivory Snow".

A: Sam Reynolds
De: Ray Jonson
Asunto: Re: ¡no seas perfeccionista!
Sam:
Tu y yo hemos tenido diferencias, pero debo decirte que personalmente me caes bien. Por favor entiende que todo lo que hago es con el propósito de SALVAGUARDAR TU TRABAJO Y EL TRABAJO DE TODOS EN ESTA DIVISIÓN. Yo te veo a ti y a toda la gente que trabajan para mi en la División de Robótica como mi familia.
Waterson fue claro: quiere tener el proyecto del robot completado a término. Y punto. Entonces, no tenemos otro recurso mas que el de "Ivory Snow". Sabes lo que quiero decir con eso. No tiene que ser perfecto. La interfaz del usuario es nuestro respaldo si esta versión del software para el robot tiene algunas fallas. El operador del robot va a estar seguro porque podrá cancelar cualquier movimiento del robot en cualquier momento. Concuerdo contigo en cuanto a que los requerimientos no funcionales son en algunas partes demasiados vagos. Lo ideal sería que si estos no fueran tiempos de apuro, cuantificáramos el tiempo que le llevaría al operador detener el robot en un caso de accidente.
Sin embargo no podemos negociar esto ahora. Como tampoco tenemos tiempo para diseñar requerimientos no funcionales nuevos y mas precisos.
No puedo enfatizar suficientemente de que estos son tiempos de apurarse. A Waterson no le cuesta nada deshacerse de toda la División Robótica. Sus amigos del Wall Street sólo le van a decir ¡Felicitaciones!. Veras, para Waterson nosotros somos tan solo del montón.
Ray

En este mensaje Ray Johnson parecía estar menos preocupado por la seguridad de comunicarse por correo electrónico.

El SENTINEL – OBSERVER entrevisto ayer por la tarde a Cindy Yardley en su propia casa. No se pudieron contar ni a Ray Jhonson ni a Sam Reynolds.

La Srta. Yardley estaba notoriamente ofuscada porque sus mensajes por e-mail fueran dados a conocer a la prensa. "De alguna forma me siento aliviada. Sentí una enorme culpa cuando ese hombre fue muerto por un robot que yo ayude a construir. Una tremenda culpa."

El SENTINEL – OBSERVER pregunto a la Srta. Yardley si es que ella había hecho una elección ética al acceder a falsear los resultados de las pruebas de software., respondió con gran emoción. "Nada, pero nada a lo largo de mi experiencia y background me preparó para algo como lo que pasó. Estudie ciencias de la computación en una universidad de gran nivel y allí me enseñaron sobre pruebas del software, ¡pero jamás me dijeron que alguien con poder sobre mi me pediría generar una prueba falseada!".

"Cuando Jonson me pidió que lo hiciera, me llamó a su oficina, como para mostrarme las trampas del poder; verá, algún día me gustaría estar en un puesto gerencial. Me senté en su oficina y vino directamente y me dijo: "Quiero que falsifiques los resultados de las pruebas de Samuels. No quiero que Reynolds se entere de nada de esto".

Yardley contuvo las lagrimas. "Me aseguró que probablemente nadie vería jamás los resultado de las pruebas dado que el robot está perfectamente seguro. Era tan solo una cuestión interna, un tema de prolijidad, en caso de que alguien en Cybernetics o de un puesto alto dentro de la corporación le diera curiosidad de ver los resultados de las pruebas. Le pregunte si estaba seguro de que el robot era seguro y todo eso y me dijo: "¡Es seguro!". La interfaz del usuario es nuestra línea de defensa. En alrededor de seis meses podemos enviar una segunda versión del software del robot y para entonces este problema de Samuels estará resuelto."

Yardley se reclinó en su asiento como si lo que dijera a continuación necesitara de un énfasis especial. "Entonces me dijo que si yo no falsificaba las pruebas, todos los de la División de Robótica perderían sus trabajos. Sobre esa base decidí falsificar las pruebas, trataba de proteger mi trabajo y el de mis compañeros."

La Srta. Yardley está al presente cursando un grado de maestría en Administración de Empresas en la Universidad de Silicon Valley.

Luego el SENTINEL – OBSERVER preguntó a la Srta. Yardley si aún sentía que había tomado una decisión ética, en vista de la muerte de Bart Matthews. "Creo que fui manipulada por Ray Jonson. El me dijo que el robot era seguro."
Otra revelación, contenida en las transcripciones del correo electrónico dadas a conocer, fue el hecho de que Randy Samuels hurto parte del software que usó en el proyecto del robot asesino. Este hecho se reveló en un mensaje que Samuels envió a Yardley cuando ella probó por primera vez su software y dió resultados erróneos:

A: Cindy Yardley
De: Randy Samuels
Asunto: Re: Maldito si lo sé
Por mi vida, no puedo entender que es lo que anda mal en esta función balancear_brazo(). Verifique la fórmula de dinámica del robot una y otra vez y pareciera estar implementada correctamente. Como sabes, la función balancear_brazo() invoca a 14 funciones diferentes. A cinco de ellas las tomé tal cual del paquete estadístico PACKSTAT 1-2-3. ¡Por favor no se lo digas a nadie! No son éstas las que causarían el problema, ¿o si?
Randy

Los expertos le dijeron al SENTINEL – OBSERVER que tomar software de paquetes comerciales de software como el PACKSTAT 1-2-3 es una violación a la ley. El software tal como el inmensamente popular PACKSTAT 1-2-3 esta protegido por el mismo copyright que protege al material impreso.

Mike Waterson, Presidente ejecutivo del Silicon Techtronics, emitió una enojosa declaración porque Max Worthington había dado a conocer las transcripciones del correo electrónico "confidencial". Las declaraciones de Waterson decían, en parte, que "Yo le pedí a nuestros abogados que intervinieran en este tema. Consideramos que esas transcripciones son propiedad exclusiva de Silicon Techtronics. Nuestra intención es efectuar cargos ya sea civiles o criminales contra el Sr. Worthington."

Como reacción a lo ocurrió ayer en el caso del robot asesinó, la ACM o Association for Computer Machinery anunció su intención de investigar si algún miembro de la ACM de Silicon Techtronics ha violado el código de ética de la ACM. La ACM es asociación internacional de computadores científicos con 85.000 miembros.

La Dra. Turina Babbage, presidente de la ACM, hizo una declaración en la Conferencia de Ciencias de la Computación de ACM que se lleva a cabo cada invierno y que esta temporada se hará en Suluth, Minnesota.

Un extracto de las declaraciones de la Dra. Babbage sigue a continuación:

"Todos los miembros de la ACM están ligados por el código de ética y Conducta Profesional de la ACM (Nota al pie: Un borrador de este código fue dado a conocer en Comunicaciones de la ACM, Mayo 1992. Por favor nótese que las declaraciones hechas por la ficticia Dra. Babbage contienen citas del verdadero código de ACM). Este código establece, en parte, que los miembros de ACM tiene el imperativo moral de contribuir con el bienestar de la sociedad y los hombres, evitar daños a terceros, ser honestos y confiables, dar crédito adecuado a la propiedad intelectual, acceder a los recursos de comunicación y de computación solo cuando así lo estén autorizados, respetar la privacidad de terceros y honrar la confidencialidad.

Más allá de eso, existen responsabilidades profesionales tales como la obligación de cumplir los contratos, acuerdos y responsabilidades asignadas, y de dar evaluaciones profundas y completas de los sistemas de computación y de sus impactos, poniendo especial énfasis en los riesgos potenciales.

Varias de las personas involucradas en el caso del robot asesino son miembros de la ACM y hay causas para creer que han incurrido en violación de código de ética de nuestra asociación. Por lo tanto, estoy solicitando al directorio de la ACM designar una Fuerza de Tareas para investigar a los miembros de la ACM que puedan haber violado groseramente el código.

No tomamos este paso a la ligera. Esta sanción ha sido aplicada sólo rara vez, pero el incidente del robot asesino no sólo ha costado una vida humana, sino que ha causado mucho daño a la reputación de la profesión de computación.

7.3.9. Articulo 9 - La revista dominical del Sentinel Observer

Una conversación con el Dr. Harry Yoder
Por Robert Franklin

Harry Yoder es una figura bien conocida en el campo universitario de Silicon Valley. El profesor de Tecnología y Ética de la Computación de Samuel Southerland ha escrito numerosos artículos y textos sobre ética y el impacto social de las computadoras. Sus clases son muy famosas, y muchos de sus cursos están completos mucho antes de que finalice el período de inscripción. El Dr. Yoder se ha recibido del Doctorado en Ingeniería Eléctrica del Instituto de Tecnología de Georgia en 1958. En 1976 recibió un grado en "Maestría en Divinidad" del Harvard Divinity School. En 1983 recibió un Master en Ciencias de la Computación de la Universidad de Washington. Ingresó en la facultad de la Universidad de Valley en 1988.
Entrevisté al Dr. Yoder en su oficina del campus. Mi intención era obtener su reacción con respecto al caso del robot asesino, y "leer su pensamiento" sobre los temas éticos que involucra el caso.

Aquí la entrevista:

SENTINEL – OBSERVER: Ir de la Ingeniería eléctrica al estudio de la religión parece un gran salto.

YODER: Yo era un Ingeniero electricista por profesión, pero todos los seres humanos tienen una vida interior, ¿no lo cree así?

SENTINEL – OBSERVER: Sí.

YODER: ¿De qué se trata su vida interior?

SENTINEL – OBSERVER: Tratar de hacer lo correcto. También se trata de lograr la excelencia en lo que hago. ¿Es eso lo que lo llevó a la Escuela de Divinidad de Hardvard? ¿Usted quería clarificar su vida interior?

YODER: Sucedían muchas cosas en la Escuela de Divinidad, y muchas de ellas eran muy poderosas. Sin embargo, más que nada quería comprender la diferencia entre lo que estaba bien y lo que estaba mal.

SENTINEL – OBSERVER: ¿Y qué hay de Dios?

YODER: Si, estudié mi propia religión cristiana y a la mayoría de las religiones del mundo, y todas ellas tenían cosas interesantes que decir acerca de Dios. No obstante, cuando yo discuto sobre ética en un foro tal como este, que es secular, o cuando discuto de ética en mis cursos de ética de la computación, no coloco a esa discusión en un contexto religioso. Pienso que la fe religiosa puede ayudarle a una persona a abrazar la ética, pero por otra parte, todos sabemos que ciertas personalidades notorias que se han autopronunciado religiosas han sido altamente no éticas. De este modo, cuando yo discuto sobre ética de la computación, el punto de partida no es la religión, sino mas bien un acuerdo común entre mis estudiante y yo de que queremos ser gente ética, que luchar por la excelencia ética es una tarea humana que vale la pena. Por lo menos, lo que no queremos es herir a otros, no queremos mentir, robar, hacer trampas, asesinar, etc.

SENTINEL – OBSERVER: ¿Quién es el responsable de la muerte de Bart Matthews?

YODER: Por favor discúlpeme si lo remito nuevamente a la Escuela de la Divinidad de Harvard, pero creo que uno de mis profesores de allí tiene la respuesta correcta a su pregunta. Este profesor era un hombre mayor, tal vez de 70 años, de la Escuela Oriental, un rabino. Este rabino dijo que de acuerdo al Talmud, una tradición antigua de la ley judía, si se derrama sangre inocente en un pueblo, entonces los líderes de ese pueblo deben ir a los límites del mismo y realizar un acto de penitencia. Esto es además de la justicia que se aplicará a la persona o personas que cometieron el asesinato.

SENTINEL – OBSERVER: Ese es un concepto interesante.

YODER: ¡Y uno de verdad! Un pueblo, una ciudad, una corporación son sistemas en que la parte esta ligada al todo y el todo a la parte.

SENTINEL – OBSERVER: Usted quiere decir que los lideres de Silicon Techtronics, tales como Mike Waterson y Ray Jonson, deberían haber asumido la responsabilidad por este incidente desde el vamos. Además, tal vez otros individuos, como ser Randy Samuels y Cindy Yardley, comparten una carga especial de responsabilidad.

YODER: Si, responsabilidad, no culpabilidad. La culpabilidad es un concepto legal y la culpabilidad o la inocencia de las partes involucradas, sean ya en lo criminal o lo civil, será decidida en la corte. Estimo que una persona es responsable por la muerte de Bath Matthews si su acción ha ayudado a causar el incidente – es una cuestión de causalidad, independientemente de los juicios éticos y legales. Las cuestiones de responsabilidad podrían serle de interés a los Ingenieros de software y gerentes, quienes tal vez querrían analizar que es lo anduvo mal, de modo de evitar que similares problemas ocurrieran en el futuro.

Mucho de lo que salió de los medios con respecto a este caso indica que Silicon Techtronics era una organización enferma. Esa enfermedad creó el accidente. ¿Quién creó la enfermedad? La gerencia creó esa enfermedad, pero también los empleados que no tomaron las decisiones éticas correctas contribuyeron con la misma.

Tanto Randy Samuels como Cindy Yardley eran recién egresados. Se graduaron en ciencias de la computación y su primera experiencia en el mundo laboral fue en Silicon Techtronics. Uno debería preguntarse si recibieron alguna enseñanza sobre ética. Relacionado a esto esta la cuestión de si alguno e ellos tenia con anterioridad experiencia en trabajo en grupo. En el momento en que se los asigno al desarrollo del robot asesino, ¿Ellos vieron la necesidad de ser personas éticas? ¿Vieron que el éxito como profesional requiere de un comportamiento ético? Hay mucho mas para ser científico en computación o ingeniero de software que tan solo la habilidad y el conocimiento de la ética.

SENTINEL – OBSERVER: Sé con seguridad que ninguno de los dos tomó cursos sobre ética o ética de computación.

YODER: Lo sospechaba. Veamos a Randy Samuels. Basándome en lo que leí en su periódico y en otros lados, era básicamente de los del tipo "hacker". Amaba la computación y la programación. Comenzó a programar en los primeros años de la secundaria y continuó a lo largo de toda su carrera universitaria. El punto importante es que Samuels aún era un "hacker" cuanto entro a Silicon Techtronics y ellos

permitieron que el siguiera siendo así. Estoy usando el termino "hacker" de un modo peyorativo y tal vez no del todo justo. El punto que estoy tratando de remarcar es que Samuels nunca maduró mas allá de su angosto enfoque como hacker. En Silicon Techtronics, Samuels aún mantuvo esta actitud en lo que hacia a sus funciones de programador, la misma que tenía cuando estaba en la secundaria. Su percepción de la vida y de sus responsabilidades no creció. El no maduró. No hay en evidencia de que tratara de desarrollarse y convertirse en una persona ética.

SENTINEL – OBSERVER: Una dificultad, en lo que hace a enseñar ética, es que en general a los estudiantes no les gusta que se les diga "esto esta bien y aquello esta mal".

YODER: Los alumnos necesitan entender que el tocar temas de ética es parte de computadores científicos o ingenieros de software profesionales.

Una cosa que me ha fascinado acerca de la situación en Silicon Techtronics es que a veces es difícil ver los límites entre lo legal, lo técnico y lo ético. Los temas técnicos involucran temas de gerencia y de computación. He llegado a la conclusión de que este desvanecimiento de los límites resulta del hecho de que la industrial del software aún se encuentra en pañales. Los temas éticos surgen abruptamente en parte porque hay una ausencia de lineamientos técnicos y legales.

En particular, no existen prácticas normalizadas para desarrollar o probar software. Hay estándares, pero no lo son realmente. Una broma muy común entre los computadores científicos es que lo bueno de los estándares es que hay mucho para elegir.

Ante la ausencia de prácticas normalizadas aceptadas universalmente para ingeniería del software, surgen muchos juicios de valor, probablemente más que cualquier otra forma de producción. Por ejemplo, en el caso del robot asesino, hubo una controversia con respecto al uso del modelo de cascada versus el de prototipo. Debido a que no había un proceso de desarrollo de software estandarizado, esto se transformó en una controversia, y los temas éticos surgen por el modo en que se resuelve la controversia. Usted recordará que el modelo de cascada no fue elegido por sus méritos sino porque el gerente del proyecto tenía experiencia en este.

SENTINEL – OBSERVER: ¿Usted cree que Cindy Yardley actuó éticamente?

YODER: Al principio su argumento parece más poderoso: ella, efectivamente mintió, para así salvar los puestos de trabajo de sus compañeros, y por supuesto, el de ella. Pero, ¿siempre es correcto mentir, para crear una falsedad, en un marco profesional?.

Un libro que usado en mis cursos de ética de la computación es el "Ethical Decisión Making and Information Technology" (Toma de Decisión Ética y Tecnología de la Información) de Kallman y Grillo.

NOTA: Este texto es un texto real y esta publicado por McGraw Hill.

En este libro se dan algunos de los principios y teorías que están detrás de la toma de decisiones. Yo uso este y otros libros para ayudar a que los alumnos desarrollen sus apreciaciones sobre la naturaleza de dilemas éticos y toma ética de las decisiones.

Kallman y Grillo presentan un método para la toma de decisión éticas y parte de su método consiste en el uso de cinco pruebas: la prueba de la mamá: ¿Le diría Ud. a su mamá lo que hizo?; la prueba de la TV: ¿Le diría Ud. a una audiencia nacional de TV lo que hizo?; la prueba del olfato: ¿lo que Ud. hizo tiene mal olor?; la prueba de ponerse los zapatos del otro: ¿le gustaría que el otro le haga lo que Ud. hizo?; y la prueba del mercado: ¿Seria su acción una buena estrategia de venta?

Lo que hizo Yardley reprobó todas estas pruebas, pienso que todos concuerdan conmigo. Por ejemplo, ¿puede imaginar a Silicon Techtronics usando una campaña publicitaria que diga algo como?: "En Silicon Techtronics el software que Usted recibirá de nosotros esta libre de bugs, porque aún cuando haya uno, distorsionaremos los resultados de las pruebas para esconderlo, Usted nunca se enterará. La ignorancia es la felicidad."

Esto demuestra que el altruismo aparente no es un indicador suficiente de un comportamiento ético. Uno podría preguntarse que otros movimientos no declarados tenía la Srta. Yardley. ¿Podría ser que la ambición personal la llevara a aceptar la explicación que le dió Ray Jhonson y su afirmación que el robot era seguro?

SENTINEL – OBSERVER: ¿Usted cree que Randy Samuels actuó éticamente?

YODER: Robar software en el modo que lo hizo es tanto ilegal como no ético.

Pienso que el punto más importante con Randy Samuels nunca fue discutido en los medios de prensa. Honestamente dudo que Samuels tuviera el conocimiento necesario para su puesto. Este tipo de conocimiento se lo llama conocimiento de la especialidad. Samuels tenía conocimiento de computación y programación pero no tenía un sólido conocimiento de la física en especial de la mecánica clásica. Su falta de conocimiento en el dominio de la aplicación fue una causa directa del horrible accidente. Si alguien con conocimientos de matemáticas, estadística y física hubiera programado el robot en lugar de Samuels, probablemente hoy Bart Matthews estuviera vivo. No tengo dudas de ello. Samuels malentendió la formula física porque no entendió su significado e importancia en la aplicación en el robot. Puede ser que la gerencia sea en parte

responsable de esta situación. Puede que Samuels les haya dicho acerca de sus limitaciones y la gerencia habrá dicho. "Y bueno, que importa".

Samuels tenía dificultades en trabajar en equipo, hacer revisiones en conjunto, y programar sin egoísmo. ¿Es posible que estuviera intentando esconder su falta de experiencia en el área?

SENTINEL – OBSERVER: ¿Cree que Ray Johnson actuó éticamente?

YODER: ¡Este tema de "Ivory Snow"! El problema con la teoría de Ivory Snow fue tan solo una racionalización para sacarse de encima a software fallado y entregarlo en término al cliente. Esta teoría solo es válida, ética y profesionalmente, si al cliente se le informa de los bugs de lo que se tiene conocimiento, o de impurezas, utilizando la jerga. En el caso de Silicon Techtronics, la teoría de Ivory Snow funcionó así: ¡sabemos que no lo es, pero al cliente hay que decirle que sí lo es!

Desde luego, presionar a Cindy Yardley como lo hizo Ray Jonson tampoco es ético. ¿El creía en lo que le dijo a la Srta. Yardley, es decir, que el robot era seguro, o fue eso una mentira del momento? Si el creía que el robot era seguro, entonces ¿por qué cubrirse con pruebas falsa? Si la interfaz con el usuario era tan importante como la última línea de defensa, entonces ¿por qué evitar pruebas más rigurosas de la interfaz?.

SENTINEL – OBSERVER: ¿Qué piensa de Mike Waterson en todo esto?

YODER: Si Jonson es el padre de la teoría de Ivory Snow, Waterson es el abuelo. Su exigencia de que el robot estuviera completado para una fecha determinada o de lo contrario "rodarían cabezas", puede haber causado que Jonson formulara la teoría de Ivory Snow. Verá, es evidente que Jonson pensaba que era imposible entregar en Cybernetics Inc. el robot CX30 para una fecha determinada, a menos que el software fuera con bugs.

En muchos sentidos pienso que Waterson actuó sin ética e irresponsablemente. Pone a Sam Reynolds a cargo del proyecto del robot, cuando aún él, Reynolds, carecía de experiencia en robot e interfaces con el usuario modernas, Reynolds rechazó la idea de desarrollar un prototipo, lo que podría haber permitido el desarrollo de una mejor interfaz.

Waterson creó una atmósfera opresiva entre sus empleados, que en sí mismo es falto de ética. No solo amenazó con despedir a todos los de la División de Robótica si el robot no se terminaba a tiempo, sino que hurgó en comunicaciones por correo electrónico privadas de toda la corporación, un derecho controvertido que algunas empresas alegan tener.

Mi creencia personal es que este tipo de investigaciones es falto de ética. La naturaleza del e-mail es algo así como un híbrido de correspondencia común y conversación telefónica. Monitorear o espiar correspondencia ajena está considerado no ético, tal como lo es interferir el teléfono. Por cierto, estas actividades también son ilegales bajo la mayoría de las circunstancias. O sea, creo que monitorear a los empleados del modo que lo hizo Waterson es un abuso de poder.

SENTINEL – OBSERVER: ¿Usted cree que en esto el fiscal tiene un caso?

YODER: ¿Contra Randy Samuels?

SENTINEL – OBSERVER: Sí.

YODER: Lo dudo, a menos que ella tenga información que hasta ahora no se ha hecho pública. El asesinato no premeditado, a mi entender, implica un tipo de acto irresponsable y negligente, que causa la muerte de un tercero. ¿Se aplica esta descripción a Samuels? Pienso que la mejor apuesta de la fiscal es hacer hincapié en su falta de conocimiento en el área de aplicaciones, si puede mostrarse que Samuels se involucró deliberadamente en un fraude.

La semana pasada leí que el 79% de la gente esta a favor de la absolución. La gente es proclive a acusar a la compañía y a sus gerentes. La otra noche, uno de los noticieros dijo: "Samuels no es un asesino, es un producto de lo que lo rodea".

SENTINEL – OBSERVER: ¿podría nuevamente decir su posición sobre el tema de la responsabilidad final en el caso del robot asesino?

YODER: En mi mente, el tema de la responsabilidad de un individuo versus la responsabilidad de la corporación, es un tema muy importante. La corporación creó un entorno en el que podían ocurrir este tipo de accidentes. Aún así, los individuos, dentro de este sistema, actuaron sin ética e irresponsablemente, y fueron los que de hecho causaron el accidente. Una compañía puede crear un entorno que saca a flote lo peor de sus empleados, pero cada empleado también puede contribuir a empeorar ese ambiente corporativo. Este es un plazo cerrado que se alimenta a si mismo, un sistema en el sentido clásico. Entonces, hay cierta responsabilidad de la corporación y cierta responsabilidad de los individuos en el caso del robot asesino.

SENTINEL – OBSERVER: Muchas gracias Profesor Yoder.