Volumen 57
Número 4 · 1983

Comunicaciones Eléctricas

Edición española de ELECTRICAL COMMUNICATION
Revista técnica publicada trimestralmente por International Telephone and Telegraph Corporation

Comunicaciones Eléctricas presenta las investigaciones, los desarrollos y las realizaciones conseguidas por ITT y sus compañías asociadas.

Publicada desde 1922 en versión inglesa, se edita actualmente en cuatro idiomas y se distribuye en el mundo entero.

Se invita a los ingenieros de ITT a proponer proyectos de artículos, cuyos resúmenes deben enviarse al editor internacional para su consideración.

Publicado en 15 de julio de 1983.

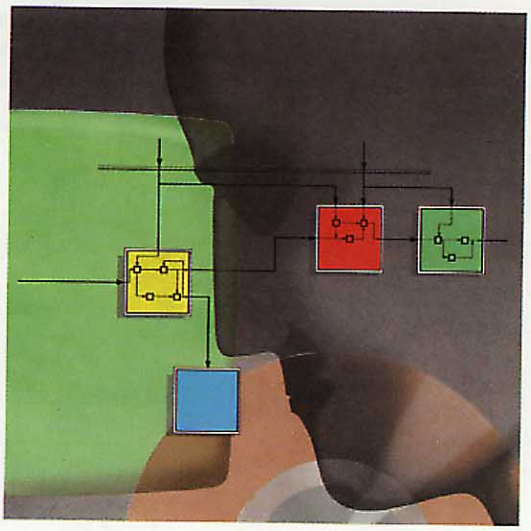
Las direcciones de los editores se dan en la página 348

Indice

Programación

Comunicaciones Eléctricas. Director en Español: A. Soto, Ramírez de Prado, 5
Madrid-7 (España)

ITT



En un programa se recoge la potencia del pensamiento humano; esta ilustración simboliza la interacción del hombre y la máquina en su producción. Al combinarse con la velocidad y precisión de la moderna microelectrónica, los programas se convierten en poderosos instrumentos que revolucionan nuestro modo de vida.

Presentación

Las tecnologías basadas en los semiconductores y en los microprocesadores son los heraldos de una era que ofrece a la humanidad una potencia de cálculo prácticamente ilimitada. La telecomunicación, el tratamiento de la información y la industria de los ordenadores, cuyas tecnologías convergen continuamente, ocupan posiciones igualmente destacadas en la carrera por aprovechar esa enorme potencia.

La programación, más que cualquiera otra tecnología, marca el ritmo de aplicación de este avance científico, al tiempo que la relativa novedad de la disciplina reclama una inversión excepcionalmente elevada en tiempo, dinero, personal y formación técnica. Además, la necesidad de innovaciones en herramientas de programación y métodos de gestión rebasa con mucho el tradicional contexto de valor añadido en el que se inscriben los productos ofrecidos por una compañía. Se necesitan notables progresos en el vasto dominio de la investigación sobre programas, técnicas y herramientas de desarrollo, procesos de producción y sistemas organizativos de control. Está en juego nada menos que la competitividad y rentabilidad de toda la Compañía. La programación es el eslabón final entre el hombre y las máquinas, si bien en ciertas actividades sea el más débil.

En la próxima década, la programación fundamentará la expansión y los beneficios de ITT; su papel es crucial en la función y sensibilidad de respuesta de productos y servicios que penetran todos los sectores de actividad, desde los tradicionales de telecomunicación y electrónica hasta los grupos de recursos naturales y productos manufacturados, pasando por panificadoras y otros servicios y artículos de consumo.

Reconociendo la universalidad de esta tecnología, ITT creó en 1978 su oficina de programación centralizada, ITT Programming, bajo cuyo liderazgo se atrajo la atención de los directivos y se dedicaron recursos a los campos de productividad, calidad e innovación en la programación. La inmediata misión de este nuevo centro fue colocar rápidamente a ITT en la vanguardia tecnológica de la programación. A más largo plazo, ITT Programming también dirige los esfuerzos de la Compañía para ensanchar el horizonte tecnológico de la era de la información.

ITT Programming coordina las actividades de programación asociadas a productos en 45 laboratorios y centros de investigación de 16 países. Su competencia en este campo la garantiza un equipo mundial de 8300 profesionales, muchos de los cuales se han formado en los centros de programación de las distintas unidades de ITT.

Al crearse en 1980 el Programming Technology and Development Center en Stratford, Connecticut (EE.UU.), ITT reunió un grupo de acreditados expertos en teoría de ordenadores, lingüística, matemáticas, proceso de datos, comunicación, gestión de la programación, psicología y enseñanza. Este equipo activo e innovador trabaja ahora en proyectos que conseguirán importantes progresos en el "estado del arte" de la programación durante los próximos 10 años, haciendo hincapié en la facilidad de uso, la reutilización y la estandarización universal.

Siendo una tecnología nueva, la programación es especialmente sensible a los cambios y a la obsolescencia. Por ello, ITT Programming orienta sus trabajos más avanzados a definir futuros entornos para desarrollo de programación que ofrezcan una amplia gama de recursos, como son: texto y gráficos integrados, herramientas de diseño y análisis de programas, lenguajes de programación de muy alto nivel, instrucciones vocales, almacenamiento y retransmisión de mensajes de voz, posiciones de programador totalmente integradas, y sistemas de enseñanza.

A más de dedicarse a investigación y desarrollo, el personal del Programming Technology and Development Center ayuda directamente a los trabajos de desarrollo de productos en las unidades ITT. Esta cooperación forma parte de la misión del Centro, que es la de presentar a todas las compañías ITT las tecnologías, herramientas y métodos de gestión más avanzados que existan en programación, a fin de que les sirvan de apoyo para el desarrollo de líneas de productos de programación de elevada calidad.

Dentro del complejo de Stratford se han formado también grupos dedicados a diversas tareas de gestión relacionadas con la programación: sistemas de planificación de productos; intercambio de información técnica; planificación y desarrollo de recursos humanos; procesos de ciclo de vida de productos; sistemas de análisis, estimación y determinación de costes; suministro separado de programas y equipos; métodos para contratos; protección de activos, y oportunidades de nuevos productos.

Por entender que la programación es el principal factor de coste en el desarrollo y mantenimiento de la central digital ITT 1240, se ha constituido el grupo de programación ITT 1240 en el International Telecommunications Center, Bruselas, con el fin de tener una visión global de las complejas actividades de desarrollo en ese área para un sistema tan innovador. Poco después se formó en Harlow, Inglaterra, el Programming Support Centre, que crearía herramientas de programación para el referido sistema.

El desarrollo de la central ITT 1240 supone la creación de un sistema básico de conmutación, dotado de un sistema de control muy especial, que ha de satisfacer la gran diversidad de exigencias de los mercados nacionales, y que debe estar "a prueba de futuro", es decir, poder incorporar ulteriores avances tecnológicos sin rediseño del sistema.

El óptimo funcionamiento del ITT 1240 – primer sistema de conmutación digital con el control totalmente distribuido – ha sido bien demostrado en Bélgica, Alemania y Dinamarca. El desarrollo del sistema operativo ITT 1240 fue el fruto de la cooperación entre unidades ITT de muchos países. Con su especialísima arquitectura de programación, el ITT 1240 es un auténtico producto multinacional y un logro destacado de toda la Compañía.

En este número de *Comunicaciones Eléctricas,* dedicado a la tecnología de la programación, se presentan algunas de las actividades de investigación y desarrollo en programación, que han estimulado la consecución del ITT 1240. Los grandes atributos de las tecnologías aquí examinadas son la garantía de calidad y fiabilidad de la programación, una mayor productividad, la claridad de la documentación y el control eficaz de la gestión de proyectos. Estas técnicas y métodos no suelen ser específicas de un proyecto concreto, sino que demostrarán su utilidad en el desarrollo de nuevos productos y servicios dentro de la variada gama cubierta por ITT. Las actividades expuestas no son sino una primera etapa en el compromiso de ITT por asumir el papel de líder en programación.

J. H. Frame
Vicepresidente y
Director de Programación de ITT

La programación en ITT

En numerosos sectores de la industria y los negocios, los productos y servicios se ven fuertemente afectados por la introducción de ordenadores (grandes, minis y micros). Siendo muchas las áreas de actividad de ITT es esencial que la Compañía tenga medios de programación con la última tecnología. Así lo reconoció ITT hace años, cuando aún estaban en su infancia los microprocesadores, adoptando una estrategia gracias a la cual sus unidades en todo el mundo disponen de completas facilidades de programación.

E. R. Kidd
ITT Programming Technology and Development Center, Stratford, Connecticut, Estados Unidos de América

Introducción

Las unidades de ITT en todo el mundo están desarrollando una gama de aplicaciones de programación más amplia que la de cualquier otra compañía. Hace tan sólo unos pocos años la gran mayoría de los productos de ITT contenía poca o ninguna programación; hoy, sin embargo, muchos de sus más importantes productos y servicios dependen de ésta, y concretamente de sus técnicas más avanzadas, para destacarse de sus competidores y garantizar el éxito.

La programación ha hecho grandes progresos en ITT desde que la Compañía decidiera asumir un importante compromiso en este área. En aquel momento, su dirección comprendió que el veloz abaratamiento del equipo de los ordenadores y de las memorias conduciría a una proliferación de aplicaciones de programación en casi todos los sectores de la industria y los negocios, y que ITT debería ocupar un primer puesto en esta revolución tecnológica.

Este artículo expone la experiencia de ITT en programación, describe sus actividades actuales en este campo y predice las tendencias futuras.

Experiencia de programación en ITT

La participación activa de ITT en la tecnología de ordenadores data de 1956, cuando se desarrolló un ordenador experimental en ITT Federal Laboratories. Un año más tarde se construyó en Europa el ordenador ITT Zebra, para uso en inventarios de almacenes. El primer sistema de telecomunicación de ITT basado en procesador fue el proyecto de centralita 480 L para el Ejército del Aire de EE.UU., en 1960. Con él se inició una serie de procesadores para telecomunicación que incluyó el ITT 1600, el ITT 3200 y sus derivados en Europa, y el ITT 640 en Norteamérica.

En los últimos años, ITT se ha esforzado por alcanzar el liderazgo en el campo de la programación. En 1976 un comité especial de la Compañía estableció la necesidad estratégica fundamental de la programación y la gran oportunidad que ésta representaba. Basándose en las recomendaciones de dicho comité, se adoptó una táctica a largo plazo. En la fase inicial se tomaron dos iniciativas básicas:

Primero, crear un centro de tecnología para apoyar a todas las unidades de ITT en áreas clave de la programación, incluyendo:

- diseño y arquitectura de componentes de programación
- facilidades de bases de datos y de comunicación de datos
- productos de soporte y herramientas de programación
- entornos de programación avanzada
- formación de programadores, gestores y ejecutivos
- transferencia de las más avanzadas técnicas de programación a las unidades de ITT.

Segundo, establecer un director de programación en cada uno de los grandes centros de ITT, responsabilizándole de las activida-

Figura 1
Distribución internacional del personal de programación de ITT, por áreas de actividad y zonas geográficas.

des de la unidad en ese campo, que comprenderían:

- consolidación de un equipo de gestión de programación
- prácticas normalizadas de desarrollo de programas
- sistemas de gestión para hallar el coste y controlar proyectos
- programas de medidas para asegurar la calidad y mejorar la productividad de los programadores
- esquemas de motivación profesional y de promociones para el personal supervisor y los programadores
- transferencias tecnológicas desde el centro primeramente citado.

El centro de tecnología y los directores de programación en todo el mundo habrían de actuar concertadamente, estableciendo una estructura del entorno de programación en todas las unidades que facilitase el desarrollo de los productos más avanzados. A mediados de 1980 el centro disponía ya del personal clave; los directores de programación iniciaron su actividad en los centros de desarrollo de productos de Europa y Norteamérica. Se creó también el Programming Support Centre en Inglaterra para desarrollar y mantener herramientas de programación, en apoyo a la central digital ITT 1240. Finalmente, se formó un equipo directivo de programación en el International Telecommunications Center, Bruselas, para dirigir las actividades de todos los programadores de dichas centrales.

Extensión de las actividades de programación de ITT

La actividad de programación de ITT es realmente internacional y se extiende a todas sus grandes áreas de interés. La figura 1 muestra la distribución del personal de programación entre sus principales campos de actividad y por zonas geográficas.

Papel de la programación en los productos y servicios de ITT

Quizá el producto de ITT más conocido, basado en programación, sea el ITT 1240, la central digital que constituye el mayor proyecto de desarrollo de ITT hasta la fecha. Su equipo de programación consta de más de 1000 personas; los programas se están desarrollando en siete centros, y el número aumenta al par que las centrales se venden en más países. El ITT 1240 reune la modularidad de programación y equipo, tecnología de integración en gran escala y microprocesadores comerciales, en centrales con control distribuido, capaces de tratar simultáneamente voz y datos. Su arquitectura de programación se diseñó según los principios económicos de los sistemas de conmutación, esto es, que una central admita el

crecimiento gradual desde unas pocas líneas hasta más de 100.000 e incorpore con facilidad los servicios de telecomunicación futuros. Su diseño y ejecución en cuanto a programas aseguran la viabilidad del ITT1240 y su capacidad de ampliación hasta bien entrado el próximo siglo.

Muchos otros productos de ITT, enumerados seguidamente, dependen en alto grado de su contenido de programación.

El ITT3150, producto Teletex de ITT, incorpora en su programación los protocolos CCITT para transmisión de documentos por redes internacionales de telecomunicación. Puede sustituir al télex, con capacidad para tratamiento de textos. Este desarrollo en lenguaje de alto nivel se basa en una moderna metodología de diseño que emplea máquinas de estados finitos; la programación se realiza en un entorno UNIX.

El ITT3030, orientado hacia el mercado europeo de microordenadores, ofrece al usuario en su arquitectura de programación una amplia gama de facilidades, incluyendo tratamiento de textos, informes financieros, paquetes de tratamiento de gráficos y sistemas de gestión de ficheros.

La posición de trabajo ITT3290 usa lenguaje de alto nivel y tecnología de microproceso. Responde a un deseo de las grandes compañías: un equipo integrado que sirva tanto de ordenador personal autónomo como de terminal para acceder a un ordenador principal.

El servicio "ITT Longer Distance" en Norteamérica constituye una alternativa económica y de alta calidad respecto al servicio telefónico público de larga distancia. Ofrece servicios entre estados (en EE.UU.) e internacional, optimizados para satisfacer las necesidades de los clientes empresariales y residenciales. Utiliza tanto microprogramación como lenguaje de alto nivel, dentro de una arquitectura lógica modular en línea con el mercado, de rápida evolución y rico en facilidades, de los "Other Common Carriers".

La familia de PABX electrónicas de ITT es la UNIMAT*. Utiliza tecnología LSI y control por microprocesador para atender las necesidades de las entidades comerciales europeas, tanto en telefonía como en una amplia gama de servicios no telefónicos. Su programación se realiza en lenguajes de alto nivel y lenguajes orientados al problema, y usa ingeniosas técnicas de simulación. Además, la necesidad de adecuar las instalaciones a los requisitos particulares de los clientes, así como de producir la central en gran escala para reducir los costes al mínimo, obligó a crear una nueva generación de sistemas de control de configuración y de desarrollo de programas.

El conmutador de paquetes de ITT permite transmitir textos en segmentos prefijados (paquetes), a través de una red telefónica pública conmutada según el protocolo X-25 del CCITT. El producto se desarrolló en un entorno UNIX, usando el PASCAL como principal lenguaje de programación.

ITT Gilfillan está desarrollando sistemas de telemando, control, comunicación e inteligencia para la Marina de Estados Unidos, que ofrecerán acceso en tiempo real a la información necesaria para el mando.

ITT Rayonier ha instalado un sistema de control por microprocesador para los equipos que transforman trozos de madera en pulpa, incorporando un visualizador con gráficos en color.

Otros ejemplos de productos y servicios ITT con importante contenido de programación son: ITT Dialcom, con el correo electrónico; Continental Baking, con un sistema para controlar el itinerario de camiones de suministro; Redshaw, que ofrece un sistema de comunicación y soporte para agencias independientes de seguros.

Además de los destinados a sus productos, ITT genera muchos programas de soporte interno para automatizar sus propias actividades, como los inventarios y nóminas. Otro tipo de programas de soporte interno se utiliza en el desarrollo de productos y servicios; por ejemplo, el *sistema de soporte para la guía telefónica* sirve para preparar las "páginas amarillas" de las guías telefónicas que publican las unidades ITT World Directories en Europa, Africa del Sur, y Puerto Rico. Consiste en una aplicación basada en VAX, que utiliza COBOL y FORTRAN con base de datos integrada para procesos por lotes e interactivos.

Finalmente, ITT ha creado un conjunto de herramientas denominado *sistema soporte para desarrollo de programas*[1], sobre el que se fundamenta el desarrollo de los programas de producto, especialmente en casos de gran complejidad como el ITT1240. Se utiliza este sistema soporte para crear, probar, depurar y mantener programas; sus herramientas incluyen:

CHILL/370, un compilador por lotes con la técnica más avanzada, desarrollado para satisfacer los requisitos del ITT1240 respecto a generación de código compacto. Su arquitectura permite transportarlo a distintas máquinas objeto, y sus algoritmos de optimización reducen el espacio del código objeto y aumentan su velocidad de ejecución.

Medios de prueba en laboratorio, un banco de pruebas basado en VAX y con lenguaje

* Marca registrada del Sistema ITT

fuente CHILL, para las pruebas de integración, calificación y regresión de las centrales ITT 1240. Ofrece entornos de trabajo por lotes e interactivo, apoyando las actividades concurrentes de pruebas.

Ejecutor de pruebas, que apoya la depuración del lenguaje fuente CHILL del ITT 1240 en un entorno simulado.

Sistema de control y distribución de cambios, base de datos en línea, interactiva, que contiene información sobre los cambios del ITT 1240. El personal puede acceder a la información desde terminales remotos,

Centro de Tecnología y Desarrollo de la Programación de ITT (PTDC) en Stratford, Connecticut.

correspondientes a las centrales instaladas en todo el mundo. Su realización se basa en un eficaz almacenamiento y recuperación de datos en diversos formatos.

Sistema soporte para control integrado de documentación, usado para crear y mantener los componentes de la documentación del ITT 1240, así como para producir documentos combinando textos y gráficos a partir de dichos componentes.

Centro de Tecnología y Desarrollo de Programación de ITT

Desde el PTDC (Programming Technology and Development Center) en Stratford, Connecticut, se ofrece un soporte centralizado a la programación en las compañías ITT de todo el mundo. El "estado del arte" evoluciona tan aprisa que un programador dijo: "si sueltas el lápiz, te retrasas en tres meses". Aunque esto sea una exageración, es innegable que se precisa un gran esfuerzo para mantener al personal de programación en primera línea tecnológica. El PTDC apoya al personal directivo de cada unidad ITT en todo lo relativo a programación, incluyendo las tecnologías utilizadas dentro y fuera de ITT, así como aquéllas de vanguardia que están aún investigándose, en desarrollo o en prototipo. Sus responsabilidades básicas descansan sobre seis grupos operativos.

El grupo de formación en programación[2] se encarga de los cursos de entrenamiento que mantienen a programadores y jefes de programación en la vanguardia de su profesión. Durante 1982 se impartieron unos 22000 alumnos-días de instrucción y de puesta al día de ejecutivos, a personas de 56 compañías de ITT. El objetivo es que cada programador de ITT participe 20 días por año en cursos de formación avanzada. El grupo realiza también inventarios de personal especializado, pudiendo así la dirección de las unidades ITT valorar la capacidad de su personal de programación.

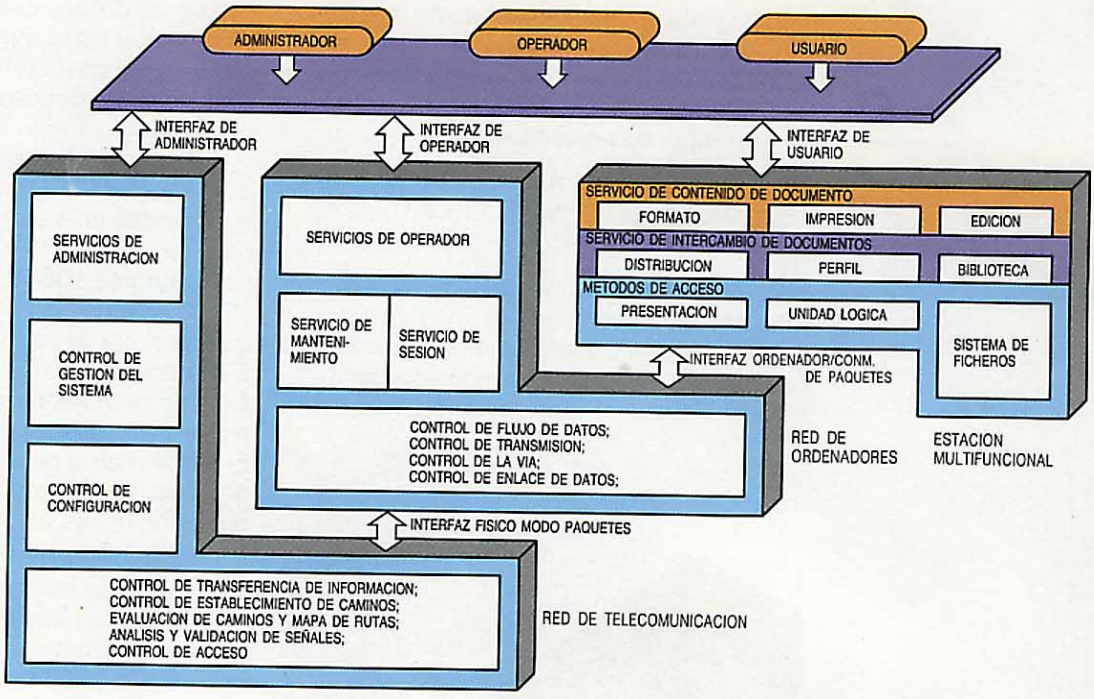
El grupo de tecnología aplicada en programación se especializa en la evaluación, selección, construcción de paquetes y normalización de métodos y herramientas de programación ya disponibles. Realiza inventarios de los recursos, evaluando la calidad de las herramientas, facilidades, controles, y procedimientos de programación en las distintas unidades. En este grupo se ha creado un sistema de medida de programación, reconocido como el método más avanzado para evaluar el rendimiento de un proyecto. Por otra parte, el programa de eliminación de defectos del ITT 1240, realizado en 1981, facilita la corrección de errores de codificación en las primeras etapas del desarrollo, sin esperar a las fases de prueba y depuración posteriores.

El grupo de tecnología avanzada de programación[3] se ocupa de cómo mejorar la eficacia de los programadores por medio de cambios en su entorno de trabajo.

El centro de sistemas de programación facilita servicios de ordenador, terminales, y otros, al personal del PTDC. También apoya a las unidades en la introducción, mantenimiento, y mejora de las tecnologías transferidas del PTDC, como por ejemplo un programa que presenta gráficamente la situación planificada y la real de un proyecto, expresada en recursos, cronogramas, costes, etc.

El grupo de arquitectura de red[4] es responsable de desarrollar estructuralmente la ITT/NET, una red interna de mensajes (Fig. 2), que dé soporte a las actividades internacionales de programación de ITT. Ello incluye la evaluación de redes de ámbito privado y de servicios de valor añadido, así como su integración en la arquitectura para soporte de una gran variedad de nuevos productos y servicios de tales características, introducidos por ITT.

Figura 2
Arquitectura global de una red de mensajes.



El centro de tecnología de sistemas proporciona soluciones y alternativas de arquitectura, diseño y realización de la programación. Las unidades de ITT pueden solicitar ayuda para resolver problemas relativos a componentes de programación tan fundamentales como programas de control, compiladores y bases de datos. El grupo está definiendo un conjunto de bloques reutilizables, constituidos por los componentes indicados, que se puedan usar en productos y servicios desarrollados por las unidades de ITT. La reutilización de los bloques constructivos redundará en una importante reducción de los costes de desarrollo y asegurará que cada producto nuevo se construye sobre una sólida base de componentes ya probados.

Técnicas de gestión de programación desarrolladas por ITT

Se han desarrollado varias técnicas nuevas de gestión de programación en las unidades de ITT y en el PTDC.

La *guía de productividad y calidad de programación* es una herramienta de gestión que mide la productividad de los programadores, la calidad del producto y los costes del proyecto. Se obtienen estadísticas de los proyectos terminados, incluyendo número de instrucciones, personas y horas empleadas, costes y defectos. Se han reunido y almacenado en base de datos estadísticas de 21 unidades de ITT y 64 proyectos, con un total de 5,9 millones de líneas de código. La herramienta es muy valiosa para evaluar el resultado de los proyectos completados y para prever los requisitos de los proyectos futuros. El estudio de los datos recogidos mostró claramente que la productividad y la calidad pueden mejorarse mediante la aplicación del diseño modular estructurado (top-down), revisiones de diseño, inspección de código, y un programa de garantía de calidad. Con estos medios para cuantificar los resultados del trabajo de programación, cada unidad podrá mejorar su rendimiento en las áreas de calidad, fiabilidad, coste de servicio y de mantenimiento, planificación y recursos humanos. Estas prácticas se han aplicado ya en los grupos de programación de ITT, colocando los cimientos para la introducción posterior de elaboradas herramientas que incrementen aún más la productividad.

El "Early Warning System" es un grupo del centro de tecnología de sistemas que informa a la dirección de las unidades sobre el estado de los proyectos con mucha programación, a fin de identificar asuntos críticos de gestión y dar directrices para la eliminación o reducción al mínimo de los riesgos. Su misión incluye asimismo cooperar con la dirección de las unidades en construir e implantar medidas correctivas. Aparte de los proyectos de ITT en curso, el grupo evalúa proyectos previstos, adquisiciones potenciales de la Compañía, proyectos conjuntos ITT/proveedores y proyectos realizados por subcontratistas de ITT. El director de unidad puede solicitar la asistencia del grupo en cualquier etapa del ciclo de vida de un proyecto, generalmente antes de comprometer recursos importantes.

La necesidad de este grupo desaparecerá cuando las unidades hayan establecido del todo el proceso de revisión del ciclo de vida de los productos (Fig. 3), realizando revisiones en sus distintas fases. Estas asegurarán que el desarrollo de un producto o servicio progresa conforme a los planes, en cuanto a coste, tiempos, contenido, productividad y calidad. En dichas revisiones participarán representantes de las funciones primarias de la unidad, entre ellas producción, marketing, financiero, servicios, para acordar totalmente todos los aspectos del desarrollo y planificar las necesarias contingencias. El proceso dará una base a la dirección para sus decisiones críticas; por ejemplo, terminar un proyecto que "falla" una fase de revisión o ampliar un proyecto que tiene éxito con nuevas mejoras a medio plazo o de la próxima generación. También evitará el tener que atender a necesidades imprevistas (ej., campaña de marketing, programa de mejoras o de mantenimiento) después de que un producto haya salido al mercado. En general, el proceso de revisión permitirá eliminar el indisciplinado método de operar que tiende a acumular atención y recursos en productos de gran éxito a expensas de los que dan buenos resultados. En la tabla 1 se resume la evolución de los productos de programación durante su ciclo de vida.

Tendencias futuras en programación

Los esfuerzos de ITT en programación se están planificando y realizando de acuerdo con varias tendencias industriales significativas.

Se está trabajando mucho en la industria de la programación para reducir los costes de desarrollo mediante la creación de módulos reutilizables, e identificación de aquéllos ya disponibles que admitan reutilización. La idea es usar componentes comprobados y que hayan demostrado su fiabilidad en más de una aplicación, en lugar de desarrollar un sistema nuevo completo en cada ocasión. Por ejemplo, pueden usarse repetidamente programas de control para microprocesadores, construyendo sobre ellos diversos programas de aplicación. De forma similar, los programas de gestión de bases de datos pueden sustentar una serie de aplicaciones desarrolladas en respuesta a nuevas oportunidades del mercado. La figura 4 muestra el impacto potencial de los módulos reutilizables en grandes sistemas de programación.

El interés por la eficacia en la eliminación de errores ha originado una segunda tendencia, persiguiendo un código exento de errores. La difusión de los ordenadores personales, pone los recursos de programación a disposición de usuarios con poca o ninguna comprensión de informática; por ello cada vez es más necesario producir sistemas de programación fiables, que se ejecuten sin error y no precisen de mantenimiento ni de correcciones. En los últimos 10 años, el número medio de defectos residuales en un programa ya entregado ha bajado desde los 10 a 20 errores por 1000 líneas de código hasta menos de uno. El objetivo de cero-defectos se conseguirá en último término mediante la tecnología de eliminación de defectos. Esta tendencia beneficia a la dirección del proyecto y al cliente: el coste de mantener un sistema después de su entrega (Fig. 5) se reduce, y

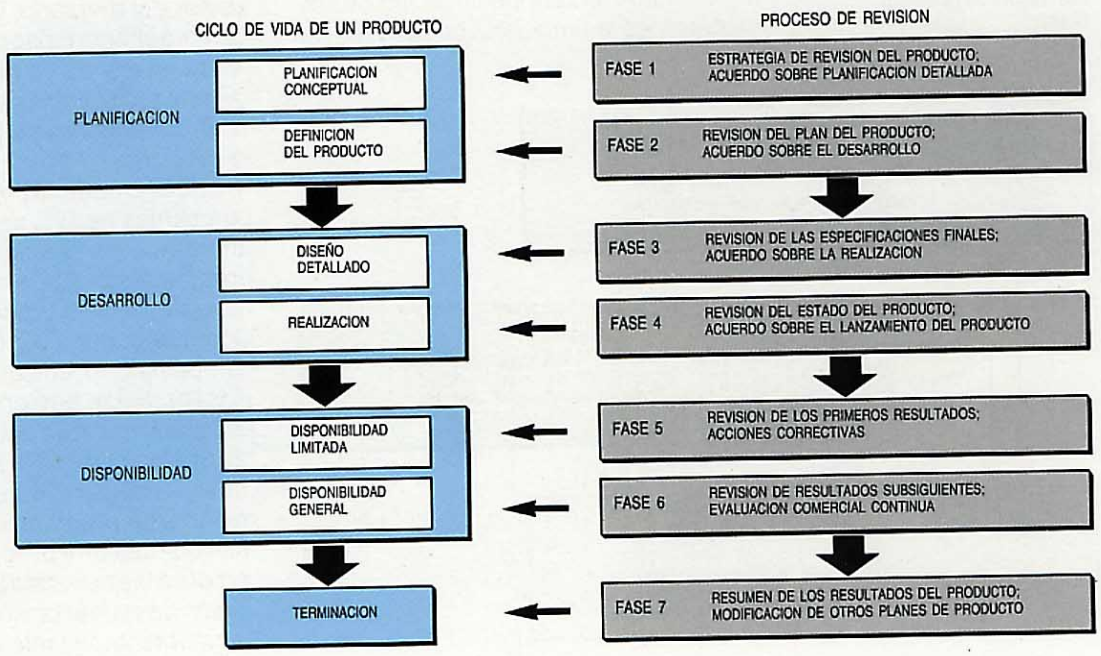


Figura 3
Proceso de revisión directiva durante el ciclo de vida de un producto; la revisión al completarse cada fase mantiene el avance del desarrollo de acuerdo con los planes.

Tabla 1 — Evolución de los productos de programación en el ciclo de vida

Actividad	Componentes de coste implicados	Representación del producto
Definición del sistema	Definición de requisitos de programas Descripción del sistema de programación Planificación del desarrollo de programación Análisis de los cambios de ingeniería	Especificaciones del sistema de programación Documento descriptivo del sistema de programación
Diseño de los programas	Diseño funcional Diseño de los programas Diseño de pruebas Herramientas de programación Evaluación del diseño	Especificaciones del diseño funcional Especificaciones de diseño de programas
Desarrollo de programas	Desarrollo de módulos Pruebas durante el desarrollo Análisis y corrección de problemas	Bibliotecas de desarrollo (módulos)
Pruebas del sistema	Procedimientos de prueba del sistema Integración de los programas y pruebas	Biblioteca de la integración del sistema
Pruebas de aceptación del sistema	Soporte de pruebas del sistema Soporte de pruebas de aceptación	Biblioteca del sistema Biblioteca de aceptación del sistema suministrado
Soporte operacional	Soporte de operación del sistema Formación Soporte de campo	
Soporte general	Gestión de proyecto Control y gestión de configuración Ingeniería de costes de programación Garantía de calidad Centros de administración y publicaciones técnicas	

la productividad de los programadores aumenta. Anteriormente, más de la mitad del personal de desarrollo tenía que permanecer en el proyecto para corregir defectos de los programas en servicio; ahora en cambio, los programadores quedarán libres para otros proyectos y oportunidades.

Otro resultado de la popularización de los ordenadores es la tendencia hacia la mayor facilidad de uso de los programas. Hasta muy recientemente, las facilidades de programación eran diseñadas por programadores y para programadores. El reto actual es hacer sistemas informáticos cuyo manejo esté al alcance del hombre de la calle. El "agrado al usuario" se ha convertido en un objetivo de la industria. Los dos factores más importantes para diferenciar los numerosos productos que aparecen en el mercado, pueden muy bien ser el grado de integración de funciones múltiples y la facilidad de uso. El éxito de un producto o servicio dependerá grandemente del nivel logrado en dichos aspectos. Por supuesto, el objetivo es incluir estos requisitos durante el diseño inicial del producto y no tratar de añadirlos en alguna etapa posterior.

Una cuarta tendencia es la denominada "unbundling": la separación de los programas y el equipo físico, y su venta como entidades independientes. La idea es tratar cada instrucción de programación como un activo de ITT. Ello significa en sí una nueva gama de productos comerciales, obteniéndose un beneficio adicional del esfuerzo de programación invertido en el desarrollo de productos que sólo tienen equipo físico.

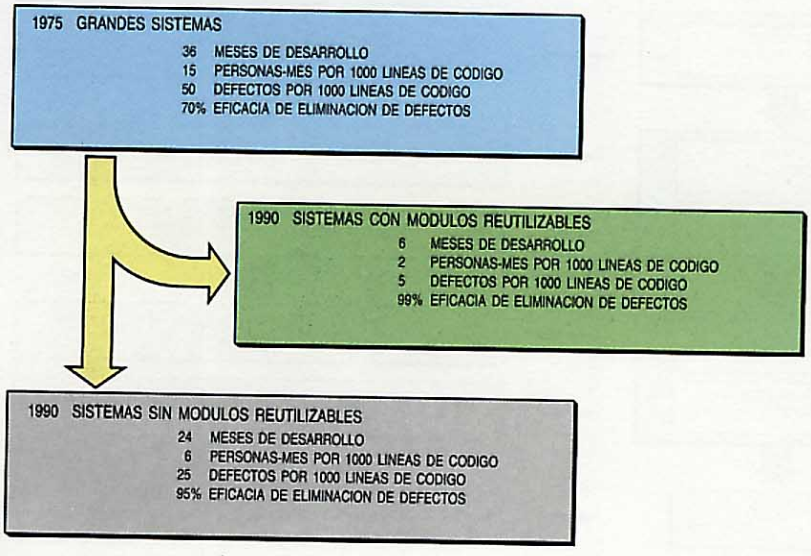


Figura 4
Impacto potencial de los módulos reutilizables.

Conclusiones: el futuro de la programación en ITT

Todas las tendencias indicadas son importantes para los productos y servicios de ITT, tanto hoy como en el futuro. La producción de la Compañía cubre todos los aspectos de la programación, con la excepción de las grandes unidades de proceso centralizado. ITT ha introducido ordenadores de oficina en Europa, y con sus compañías Qume y Courier ocupa una buena posición en este sector del mercado. Otras unidades producen ordenadores personales, aplicaciones financieras y de seguros, productos de defensa, y sistemas de tiempo real. Todas ellas continuarán mejorando su posición en la industria.

El PTDC seguirá prestando su apoyo a las compañías de ITT, para mantenerlas al día en las nuevas prácticas y tecnologías de programación. El Centro también se ocupará de desarrollar productos de programación que puedan ser incorporados por las compañías ITT en sus propios productos.

Es probable que en el futuro aumente la especialización de actividades en las distintas unidades de ITT: una de ellas puede prestar una atención especial a las herramientas de programación, otra a los sistemas operativos y programas de control, una tercera a las actividades de diseño y optimización de redes. Lo que se intenta con esto, naturalmente, es minimizar la duplicación

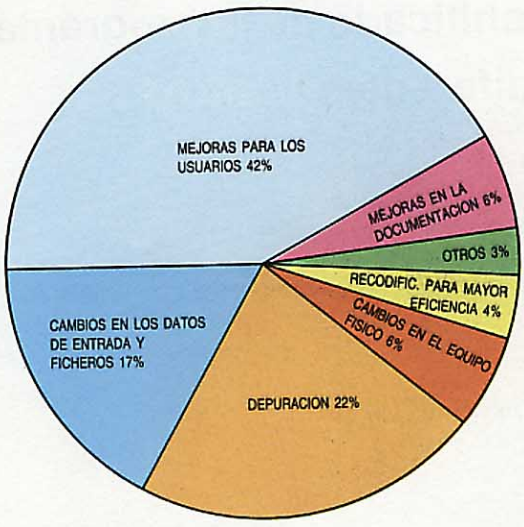


Figura 5
Funciones de mantenimiento después de la entrega del producto.

de esfuerzos y así reducir los costes totales de desarrollo de los productos. El PTDC actuará como punto focal de estos esfuerzos, coordinando la transferencia de tecnología entre todas las compañías ITT.

El futuro de la programación en ITT depende de conseguir el mejor equipo profesional y de gestión, y la mejor calidad, productividad y facilidad de reutilización de sus programas. Para lograr estos objetivos en algunas unidades ya existen, y en otras se están preparando, medios de gestión, medición, formación y transferencia de tecnología. La evolución de la programación en ITT hacia una posición de liderazgo en la industria se encuentra en franco avance.

Referencias

1 M. P. Saward: Herramientas para soporte de la programación: *Comunicaciones Eléctricas,* 1983, volumen 57, nº 4, págs. 289–294 (en este número).
2 H. F. Moody y F. Wilson: Entorno de aprendizaje interactivo para enseñanza de la programación: *Comunicaciones Eléctricas,* 1983, volumen 57, nº 4, págs. 315–319 (en este número).
3 A. W. Holt, H. R. Ramsey y J. D. Grimes: La tecnología de sistemas de coordinación como base para un entorno de programación: *Comunicaciones Eléctricas,* 1983, volumen 57, nº 4, págs. 307–314 (en este número).
4 S. R. Kimbleton y P. S.-C. Wang: Soporte de comunicación ITT/NET para el desarrollo de programas: *Comunicaciones Eléctricas,* 1983, volumen 57, nº 4, págs. 320–325 (en este número).

Earl R. Kidd nació en Cleveland, Ohio en 1938 y recibió el grado BS en estadística matemática en la Purdue University. Durante 18 años trabajó en IBM, ocupando puestos en dirección y desarrollo de productos, hasta alcanzar el de Corporate Director of General Purpose Systems, siendo responsable de todos los productos de equipo y programación para los grandes sistemas de IBM. En 1979 entró en el PTDC de ITT como director de desarrollo y tecnología de programación, con total responsabilidad sobre el desarrollo de recursos y la capacidad de programación de ITT en todo el mundo.

Desarrollo planificado de la programación ITT 1240 por diferentes centros

La estructura de control distribuido de la central digital ITT 1240 se presta al desarrollo en unidades de programación geográficamente distantes, si bien han de cuidarse la partición, los interfaces y la oportuna edición de los cambios.

D. A. Lawson
International Telecommunications Center, Bruselas, Bélgica

Introducción

La central digital ITT 1240 con su arquitectura de control distribuido es el resultado de un desarrollo repartido entre equipos de Alemania, Bélgica, España, Estados Unidos e Italia, que trabajan juntos bajo la dirección del International Telecommunications Center (ITC) en Bruselas. La idea fundamental del diseño de la estructura de control distribuido del ITT 1240 consiste en dotarle de flexibilidad para atender necesidades de mercado muy diferentes y para incorporar mejoras futuras de equipo y de programación (Fig. 1). Interesa destacar que el mismo método de desarrollo de la programación proporciona ambas ventajas: flexibilidad para cumplir casi todos los requisitos específicos de mercado, incluyendo adiciones posteriores, y capacidad para acomodarse a los más recientes avances tecnológicos, asegurando que el sistema ITT 1240 será inmune al futuro.

Como las compañías de desarrollo y fabricación de ITT están situadas en muchos países, el ITT 1240 fue diseñado de modo que cada compañía nacional pudiera básicamente ser autosuficiente en la fabricación de centrales y en el desarrollo de los programas para cumplir sus requisitos nacionales y de exportación. La magnitud del proyecto ITT 1240 (del orden de varios miles de ingenieros/año, más de cuyos dos tercios se dedicaron al desarrollo de los programas del sistema y de aplicación) impidió que ninguna compañía nacional desarrollara el sistema por sí sola.

Ventajas del desarrollo distribuido

El desarrollo distribuido del sistema ITT 1240 se traduce en varias ventajas importantes para una multinacional como ITT. Estas son:

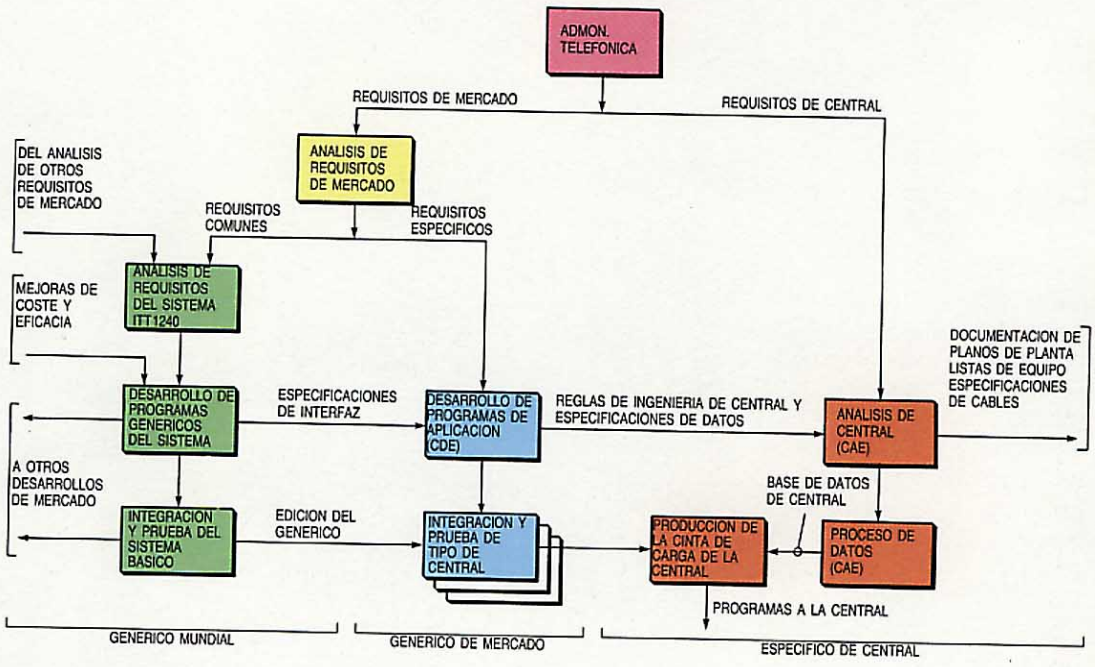


Figura 1
Proceso de los requisitos de mercado.
CAE - ingeniería de aplicación
CDE - ingeniería de diseño.

- *Transferencia de tecnología;* puesto que las compañías nacionales participan en el desarrollo del sistema, asimilan tecnologías de vanguardia en todo momento.
- *Intercambio de tecnología;* el desarrollo multinacional significa que las compañías nacionales comparten sus tecnologías y su saber, ampliando así su base de conocimientos.
- *Evaluación mundial de las tendencias y mercados de telecomunicación;* el equipo de diseño ITT 1240 estudió el estado actual de la telecomunicación y la probable evolución de la tecnología. Las compañías nacionales aportaron información sobre sus presentes exigencias de mercado, y previsiones sobre futuras demandas de servicios. La evaluación de las tendencias y orientación del mercado condujo a la decisión de crear una central digital con el control totalmente distribuido. Las previsiones de requisitos de mercado están a disposición de cada una de las compañías de ITT para su planificación nacional y de exportación.

El desarrollo distribuido de los paquetes de programación del ITT 1240 suscitó aspectos nuevos en la gestión de un producto. El desarrollo concurrente por equipos multinacionales en múltiples lugares reclama un extraordinario esfuerzo para normalizar los métodos de producción de programas. Hay que llegar a un acuerdo sobre el contenido de los paquetes, cómo deberían ser sus interfaces, cómo deberían probarse y cómo tendrían que incorporarse e integrarse con los programas previamente instalados. Al mismo tiempo, es esencial no comprometer la flexibilidad del ITT 1240 por la exigencia de normalización, y que los productos de programación permitan a cada compañía nacional programar el ITT 1240 para atender necesidades de sus clientes.

Desarrollo concurrente de programas

La necesidad de dejar actuar a cada compañía según su mercado, contribuye a sustentar el desarrollo concurrente de programas. Muchos de los programas del ITT 1240 están proyectados para uso en todos los mercados: estos programas básicos del sistema proporcionan un entorno de proceso a prueba de fallos para programas de aplicaciones de conmutación destinados a satisfacer requisitos específicos de clientes. El paquete total de una central consta, pues, del sistema básico y de los programas de aplicación, más los datos que describen la central.

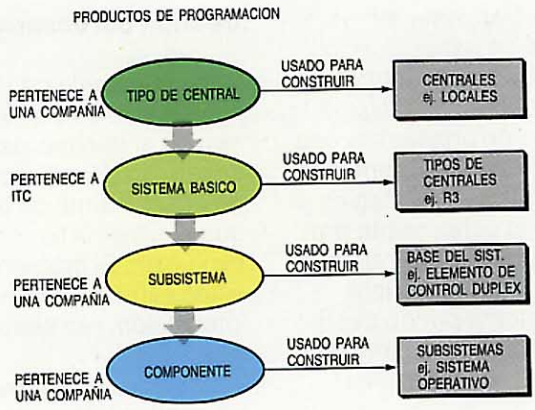


Figura 2
Jerarquía de programas ITT 1240.

Si a cada compañía se le dan programas estables del sistema básico y especificaciones de los interfaces, el desarrollo concurrente de los programas de aplicación puede realizarse con pequeñas interacciones. Si se congela el sistema básico, el desarrollo concurrente no presenta ningún problema. Esto, sin embargo, viola el principio de diseño para el cambio que el ITT 1240 se propone alcanzar; en efecto, el sistema básico necesita cambiar mientras se están desarrollando aplicaciones, con el fin de proporcionar mejoras de coste y de eficacia, así como nuevas funciones que hagan más competitivo el producto. El sistema básico es demasiado grande para ser dirigido por una sola compañía, y hay normalmente actividades de desarrollo que se solapan y que necesitan progresar a un cierto ritmo, por lo que se necesita una partición del sistema básico en productos que se puedan planificar y desarrollar concurrentemente. Esta partición se consigue del siguiente modo:

Los programas ITT 1240 se organizan en una jerarquía de cuatro niveles de productos (Fig. 2), relacionados de modo tal que un miembro de un nivel se compone de un subconjunto de miembros del nivel inferior. Los niveles son: tipo de central (local, interurbana, internacional, etc.), sistema básico y de aplicación, subsistema, y componentes de subsistemas.

Las compañías de ITT cuentan con el desarrollo de programas de aplicación para satisfacer los requisitos de los clientes en cuanto a tipos de central. Estos son genéricos para todas las centrales de un mercado. La adaptación requerida para una central particular se consigue por medio de especificaciones y asignación de datos a ese tipo de central. Se pretende que el sistema básico sea genérico para todos los mercados del mundo.

Esta división de la programación en cuatro niveles, por sí sola no hace posible el

desarrollo concurrente. Para éste el requisito clave consiste en *especificar los interfaces antes del desarrollo y controlar los cambios de interfaz durante el mismo.* Sin planificación ni partición previas hay una gran probabilidad de error en los interfaces. Los interfaces incorrectos son inestables y conducen a un desarrollo concurrente muy costoso, si no inútil, especialmente cuando se asienta en múltiples lugares. La integración y prueba del sistema puede ser un proceso interminable, porque los cambios se propagan a través de interfaces inestables.

Interfaces normalizados

La construcción sobre un interfaz predefinido y acordado se ha empleado con éxito durante años en el desarrollo de equipos; también se ha utilizado en el desarrollo de programas, aunque no con el mismo acierto.

Casi por definición, los interfaces limitan la flexibilidad de cambio. Para que un interfaz sea aceptable, hay que poder definir la mayor parte de sus requisitos con antelación y prepararlos en las especificaciones iniciales del interfaz.

La primera y segunda generación de sistemas de conmutación controlados por programa almacenado tienen recursos limitados de memoria y capacidad de proceso, debido a sus arquitecturas de control centralizado. Los programas para estos sistemas se ven así obligados a trabajar con restricciones. Además, es muy difícil prever necesidades de recursos con antelación al desarrollo, y por ello se cometen errores de previsión que se resuelven con una nueva partición de problemas y cambios de interfaces. En resumen, la limitación de recursos dificulta la definición de interfaces estables.

En el pasado el desarrollo de equipos no se ha visto tan afectado por la escasez de recursos, lo que ha permitido especificar y controlar los interfaces razonablemente bien. Sin embargo, esto empieza a cambiar con la llegada del LSI debido a exigencias de disipación de calor, espacio y asignación de terminales.

La arquitectura de control distribuido ITT 1240 ha eliminado la mayoría de las limitaciones de recursos de programación en cuanto a capacidad de proceso y memoria. Ello facilita la definición de interfaces estables y que permitan el desarrollo concurrente.

Gestión del desarrollo concurrente

Cada compañía de ITT depende de interfaces estables con el sistema básico para reducir al mínimo las interacciones con el desarrollo de aplicación. Sin embargo, son de desear cambios en el interfaz, puesto que el sistema base está en constante evolución. El problema es cómo conciliar estos cambios con niveles mínimos de interacción, rediseño y pruebas de regresión.

La solución elegida consiste en identificar los cambios del sistema básico lo antes posible, consolidarlos en grupos de trabajo afines y planificar ediciones de grupos de cambios a intervalos espaciados, sin consentir un continuo trastorno por el incesante goteo de cambios individuales. Se pretende distanciar en un año las ediciones sucesivas y, como norma formal, no menos de seis meses.

Quizás un bosquejo del análisis y partición del sistema ITT 1240 en unidades más pequeñas y manejables aclare cómo se traducen los conceptos en productos acabados.

Se mencionó ya el primer nivel conceptual de diseño, para el desarrollo concurrente de programas de aplicaciones: el tipo de central viene determinado por los programas de aplicación junto con el sistema básico, definiendo este último por sus interfaces formales con aquellos programas y por las funciones necesarias universalmente (Fig. 3).

El segundo nivel de diseño se refiere al desarrollo concurrente del sistema básico: se divide éste en subsistemas, cada uno de los cuales es un conjunto de programas del sistema para uno o más elementos de control ITT 1240. Los interfaces de un subsistema pueden verse como una vía que admite comunicaciones entre programas de aplicación y de sistema, y entre los subsistemas que configuran el sistema básico (Fig. 4).

Algunas funciones son comunes a varios tipos de subsistemas y sería antieconómico desarrollar y mantener dichas funciones por separado para cada subsistema. Para evitar este desarrollo redundante, introducimos un tercer nivel conceptual de diseño (Fig. 5): división de los subsistemas en componentes comunes (grupos de funciones afines comunes a varios subsistemas) y componentes específicos (grupos de funciones afines propias de un solo tipo de subsistema). Los interfaces de un componente común pueden considerarse como una vía de comunicaciones que permite la relación entre él y otros componentes del subsistema. La gestión del concepto de vía de comunicaciones amplía el desarrollo

Figura 3
Primer nivel conceptual de diseño: desarrollo concurrente de programas de aplicación a mercados.

concurrente hasta el nivel inferior de componente.

Pertenencia de productos de programación

Cuando un centro de diseño de ITT desarrolla un componente, adquiere la propiedad del mismo. La propiedad es indivisible en el sentido de que el propietario tiene plena responsabilidad (y la correspondiente libertad) para el diseño, desarrollo y prueba de tal componente. Otras compañías de ITT pueden utilizar dicho componente, pero no pueden cambiar su contenido. Lo mismo sucede en otros niveles de productos de programación ITT 1240 disponibles: los subsistemas, sistema base, y tipos de central son de propiedad indivisible.

Respuesta a necesidades de mercado

Aunque la sucesiva división del sistema básico en subsistemas y después en componentes proporcione el entramado para los cambios, lo que se necesita es un plan para organizar y dirigir el trabajo de desarrollo de programas, de manera que responda a los requisitos del mercado.

La ingeniería de diseño (CDE) cumple los requisitos específicos de un mercado nacional, y consiste en el desarrollo de programas de aplicación mediante un conjunto normalizado de herramientas y procedimientos[1]. Los datos específicos de una nueva central se dan a través de la ingeniería de aplicación (CAE), que toma información suministrada en parte por la Administración y la procesa para producir la base de datos inicial. Si bien las CDE y CAE proporcionan una parte específica de los requisitos del cliente, lo esencial del producto entregado proviene del sistema básico, que contiene los requisitos genéricos o comunes del sistema.

Necesidad de una base genérica

Cuando ITT se embarcó en el desarrollo de la central digital ITT 1240, se dispuso a crear un sistema básico único, que no solamente cubriera una amplia variedad de requisitos de diferentes países, sino que pudiera también incorporar futuras mejoras tecnológicas sin rediseños en el sistema. La necesidad de una base genérica para su programación es resultado directo de la decisión de desarrollar un sistema digital flexible para todas las aplicaciones.

Figura 4
Segundo nivel conceptual de diseño: desarrollo concurrente de programas del sistema.

Una vez identificados los elementos comunes del sistema, sus costes se pueden reducir apreciablemente mediante el desarrollo de programas para todas las aplicaciones a la vez, en lugar de hacerlo para cada aplicación. Esto significa menor trabajo de diseño, desarrollo, prueba y mantenimiento, y un producto más fiable para el cliente. Por ello se necesita una base genérica, tanto por técnica como por economía.

Debido a que el desarrollo del sistema base genérico es demasiado extenso y complejo para una sola compañía, se divide la base en subsistemas y componentes menores, individualmente producidos por equipos de diseño en diferentes compañías. Durante este proceso, los miembros de los distintos equipos han acumulado experiencia en las más avanzadas técnicas de desarrollo de programas, lo que constituye un haber valioso. La propiedad de componentes, subsistemas y programas de aplicación es de las compañías que los han desarrollado.

El plan general para producir el sistema genérico se ilustra en la figura 6. El ITC define la parte común del sistema, distribuye el trabajo entre los centros de diseño, y coordina y planifica su ejecución. Cuando se requiere una nueva edición del sistema básico, la integración se hace en uno de los

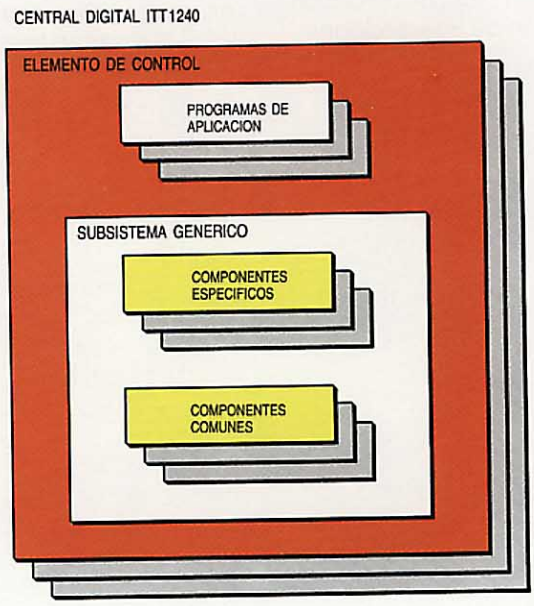


Figura 5
Tercer nivel conceptual de diseño: evita esfuerzo redundante.

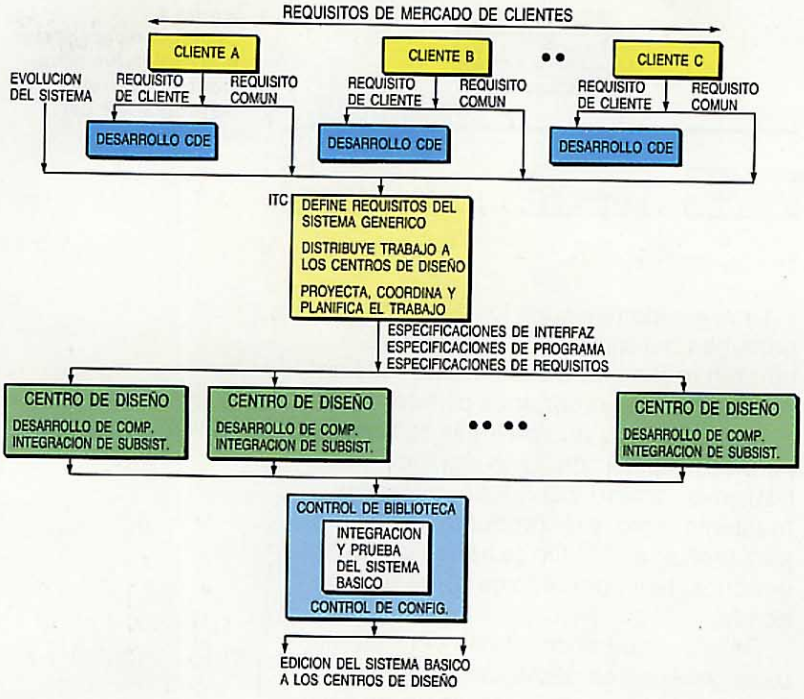


Figura 6
Diseño y desarrollo del sistema genérico.

centros de diseño, controlando el ITC la biblioteca o la configuración.

Conclusiones

El ITT 1240 se ha diseñado para adaptarse a los mercados mundiales de conmutación y para ser inmune al futuro, objetivos que requieren una estrategia de gestión de programación capaz de hacer frente con eficacia a cambios constantes. Además, el desarrollo concurrente de programas en diferentes lugares del mundo crea una serie de problemas en dicha gestión de programación. Afortunadamente, la arquitectura de control distribuido ITT 1240 facilita el desarrollo descentralizado de programas. Esta estrategia se ha hecho realizable gracias a un gran cuidado en la partición, planificación de interfaces y limitación de interacciones mediante edición controlada de cambios, reduciendo así al mínimo los problemas de la gestión.

Referencia

1 M. P. Saward: Herramientas para soporte de la programación: *Comunicaciones Eléctricas,* 1983, volumen 57, n° 4, págs. 289–294 (en este número).

David A. Lawson ingresó en Bell Laboratories en 1963, donde trabajó en diseño de tratamiento de llamadas y en integridad de programación para un sistema de conmutación electrónica. Después pasó a ser responsable de metodologías de desarrollo para ampliaciones de equipo y programas en sistemas de conmutación electrónica en servicio. Posteriormente se encargó del desarrollo de herramientas de documentación, métodos y arquitectura para nuevos programas. En 1976 ingresó en ITT como director adjunto para el diseño de herramientas soporte en lenguajes de alto nivel y programas operacionales. Tras haber sido adjunto al director de arquitectura de sistemas avanzados en el ITT PTDC, el Sr. Lawson trabaja ahora en el ITC, Bruselas.

Herramientas para soporte de la programación

Los sistemas soporte y las herramientas de programación son esenciales para el desarrollo de cualquier producto que tenga un alto contenido de programación. Pueden mejorar la fiabilidad de los programas y asegurar que el desarrollo sea de un coste adecuado y se termine a tiempo. Con un diseño cuidadoso, las herramientas de soporte pueden usarse en proyectos sucesivos con un notable beneficio económico.

M. P. Saward
ITT Europe Programming Support Centre,
Harlow, Inglaterra

Introducción

Cuando una compañía internacional acomete el desarrollo de un sistema con gran contenido de programación, ya sea un sistema de proceso de datos convencional, un control de comunicaciones y puesto de mando militar, o una central telefónica digital, se ve obligada a invertir en sistemas soporte y herramientas que faciliten la generación de programas fiables, económicos y terminados a tiempo.

En su papel de ayudar a la productividad, las herramientas de programación justifican la inversión hecha en ellas por los ahorros que consiguen al automatizar muchos de los procesos involucrados en el desarrollo, fabricación e integración de los productos de programación. Los procedimientos y procesos manuales para la construcción de programas son largos y propensos a error, en particular para sistemas complejos. La automatización de los procesos con herramientas de soporte bien diseñadas puede garantizar la calidad de los programas.

Además, una vez desarrolladas las herramientas idóneas, se pueden realizar con mayor rapidez cambios que respondan a nuevas exigencias del usuario. Un conjunto integrado de herramientas de programación puede dar a una compañía varios años de ventaja sobre los competidores que no dispongan de tales medios.

En muchos casos, las herramientas se pueden reutilizar en nuevos proyectos con importantes ahorros en tiempo de desarrollo y coste del producto, así como en la oportunidad y el coste de disposición de las herramientas. Un factor esencial para su reutilización es que tales herramientas correspondan a un gran sistema de ordenadores estándar, independiente del producto a desarrollar. Esto proporciona un entorno estable para los procesos de trabajo a soportar. Además ofrece una reserva importante de potencia y almacenamiento, y permite la creación de herramientas para varios años sin el gasto de una continua reelaboración. En el caso de ITT, se emplean los sistemas compatibles IBM 370/MVS.

En los últimos años ITT ha potenciado el desarrollo de programas incorporados en sus productos. Por ejemplo, el Unimat 4080* PABX (centrales automáticas privadas), que contiene más de medio millón de instrucciones, necesitó un esfuerzo de desarrollo de varios centenares de años-hombre. El desarrollo de programas para el sistema digital ITT 1240 a lo largo de la vida del producto superará al anterior en un orden de magnitud.

Reconociendo la necesidad de un grupo centralizado que dirigiera y coordinara el desarrollo de las herramientas soporte de programación, ITT estableció en marzo de 1981 el Programming Support Center (PSC) en Harlow, Inglaterra. La tarea inicial emprendida por el PSC fue la de completar la entrega del conjunto de herramientas para el desarrollo y fabricación del sistema ITT 1240. Esto exigió la consolidación de las responsabilidades de dirección, incluyendo la planificación, seguimiento, integración, envío e instalación de las principales herramientas, en 10 grupos geográficamente dispersos que soportaban entonces en 5 centros el desarrollo del sistema ITT 1240, así como la participación en el desarrollo de las propias herramientas. El PSC también fue responsable de definir los interfaces entre herramientas para asegurar que se conseguía la coherencia arquitectónica (y por tanto su correcta interacción).

Debido a una distribución tan amplia del desarrollo de herramientas entre los grupos

* Marca registrada del Sistema ITT

de Europa y América, es fundamental que los grupos multinacionales acuerden métodos de producción y normas de realización comunes. Si se quiere evitar una costosa duplicación de trabajo es muy importante coordinar el establecimiento de normas y procesos, lenguajes, y entornos de ordenador. Aun habiendo aceptado las normas y disponiendo de un extenso conjunto de herramientas en los centros nacionales, las variables exigencias del usuario y la evolución tecnológica obligarán siempre a armonizar los desarrollos de mejoras.

La dirección concertada global de las actividades de elaboración de herramientas requiere una estrecha cooperación con los grupos de desarrollo de productos en los diferentes centros nacionales de ITT. Es esencial saber apreciar las necesidades y prioridades de desarrollo en cada centro.

Como las herramientas soporte del sistema ITT 1240 están casi terminadas, el objetivo pasará a ser la aplicación de las mismas al desarrollo de otros programas y proyectos, a fin de maximizar el beneficio de la inversión realizada por ITT desde 1978.

Actividades de programación a soportar

Para un gran sistema como el ITT 1240, la producción y desarrollo de programas es el resultado de una serie de procesos y subprocesos discretos que se ejecutan de acuerdo con una secuencia predefinida. Dado que los programas los crean seres humanos, estos procesos y subprocesos, y su interrelación, están documentados con reglas y directrices que se recogen en manuales (instrucciones y prácticas de programación) destinados a los ingenieros, de acuerdo con las tareas especializadas que ellos realizan. Fundamentalmente, los sistemas y herramientas soporte automatizan tales reglas y directrices.

En el más amplio sentido, el objetivo del trabajo que los procesos y las herramientas realizan es el de permitir el *desarrollo* de unidades de programación (análogas a las piezas de un equipo) probadas y validadas, que pueden luego almacenarse en cierto lugar de una biblioteca de programas mediante una actividad de *producción* de programación, y posteriormente ser recuperadas de ese lugar y *fabricadas* para obtener un sistema completo (ejemplo: el sistema ITT 1240).

Es de hacer notar que estas tres distintas actividades para conseguir el producto final las realizan grupos separados con conocimientos muy diferentes. Esto tiene un impacto considerable en el diseño de herramientas, especialmente en el área de factores humanos (formatos de pantalla, restricciones de funcionalidad, validación, y control de errores).

Las actividades del sistema ITT 1240 que hasta ahora se han beneficiado de las herramientas soporte son:

- dirección del proyecto
- diseño del sistema
- diseño de los programas
- codificación y prueba
- integración
- prueba del sistema
- producción, fabricación y distribución
- documentación
- mantenimiento
- control de cambios.

Se desarrollaron paquetes soporte específicos para la producción de programas y la dirección de actividades. En resumen, las características de los principales sistemas soporte del ITT 1240 son[1]:

SDSS (sistema soporte para desarrollo de programas): utilizado por diseñadores del sistema, programadores de desarrollo, y programadores de ingeniería de clientes para crear modulos de programas cargables

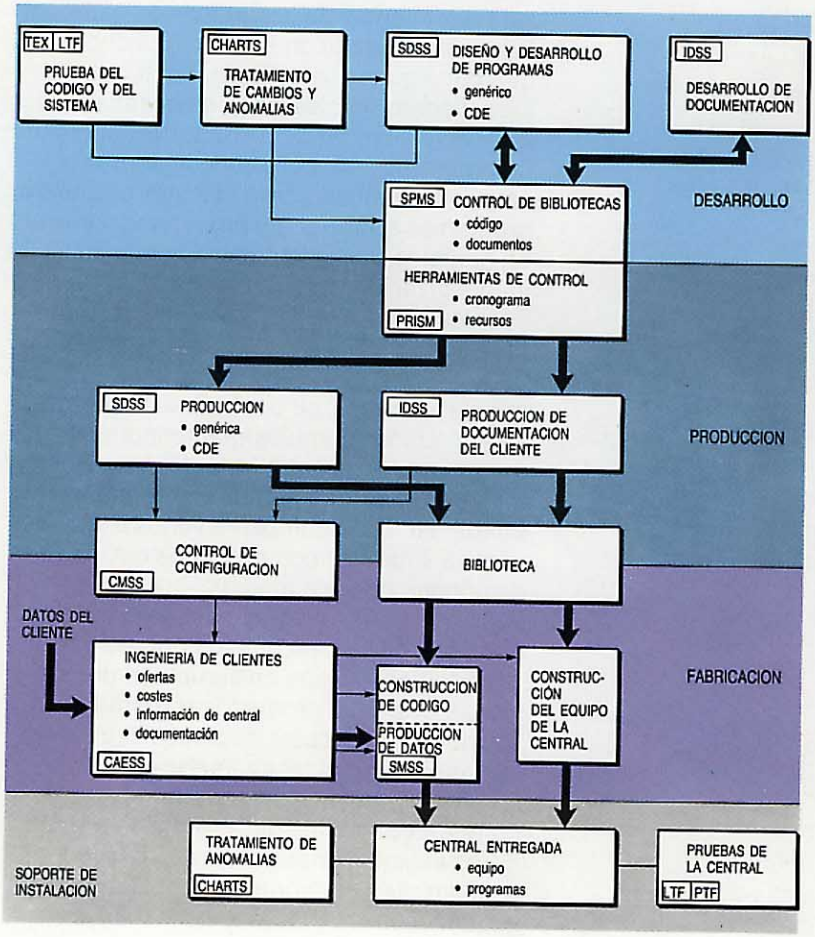


Figura 1
Diagrama general de los sistemas soporte de la central digital ITT 1240.
CDE - ingeniería de diseño para clientes.

a partir de código fuente. También se emplea en la producción automática de programas. Este sistema soporte se había utilizado previamente en otros dos productos importantes.

IDSS (sistema soporte para control integrado de documentación): usado para el desarrollo, control y mantenimiento de documentos (texto y gráficos), y para la producción de la documentación del cliente.

CHARTS (sistema de control y distribución de cambios): se aplica para el seguimiento del estado de los problemas y de la eliminación de faltas, dentro de los centros ITT o entre dichos centros. También se usa como sistema de control para la gestión ordenada de las actividades de cambio.

TEX (programa ejecutivo para pruebas): ofrece un entorno simulado del sistema/producto en el sistema de ordenador, no exigiendo un acceso inmediato al producto físico en operación. Por medio de modificaciones y órdenes en lenguaje de alto nivel consigue probar y depurar programas individuales o conjuntos de programas.

LTF (medios de prueba en laboratorio): usado para la prueba de centrales completas, o de funciones complejas de una central que se ejecutan sobre el propio equipo activo. Hay una versión de TEX que contiene también algunas de estas facilidades.

SPMS (sistema monitor de producción de programas): permite controlar las sucesivas versiones de los componentes de programas a medida que se van desarrollando, y el almacenamiento de todos los descriptores de datos usados en el producto. En especial, el SPMS ofrece al ingeniero un interfaz en modo menú a través de pantalla que le evita la complejidad propia del sistema soporte.

PRISM (sistema de control de información y estado de programas): se aplica a la planificación y seguimiento del desarrollo de programas individuales (por ejemplo, cronogramas, recursos, tamaños) y su integración (relaciones, dependencias) en programas mayores.

CMSS (sistema soporte para control de configuración): mantiene los listados maestros (equipo, programas, documentos) de la configuración de una central, y controla todos los elementos producidos para uso en la fabricación del sistema ITT 1240.

CAESS (sistema soporte de ingeniería de aplicación): acepta datos del cliente para una central específica y da una primera información de su coste para posibles ofertas; cuando el pedido resultante se procesa, produce instrucciones y especificaciones detalladas para la fabricación del equipo de la central y de los programas.

SMSS (sistema soporte para producción de programas): construye el paquete de programas de una central a partir de las especificaciones proporcionadas por el CAESS. Usa elementos de programas obtenidos de las bibliotecas, y los combina con los datos que definen la central.

Herramientas de acoplamiento

En la figura 1 se muestra el papel de los sistemas soporte en las actividades y procesos de programación del sistema ITT 1240. Es importante hacer constar que muchas de las herramientas están involucradas en más de una actividad de usuario; así, pues, el compilador CHILL en el SDSS, usado por los diseñadores de programas, se emplea también de una forma automática y sencilla en el proceso de producción por los integradores del sistema.

La actividad de desarrollo incumbe a programadores muy expertos, dedicados al diseño y desarrollo de los módulos de programas y a la documentación asociada para el producto.

La actividad de producción corresponde a los constructores del sistema, quienes deben preocuparse de que los programas individuales se creen de una forma compatible y puedan unirse en configuraciones básicas de producto (p.ej., gran central interurbana, concentrador).

La actividad de fabricación la realizan, separadamente para cada central, expertos en las exigencias y los datos de los clientes, quienes son responsables de la entrega final de la central.

La producción de estos elementos es una tarea compleja. Miles de unidades de programas, muchos de ellos semejantes, deben estar disponibles a tiempo y ser compatibles entre sí cuando se está fabricando una central. Se requieren herramientas soporte para la planificación y el seguimiento de las diferentes actividades, así como para mantener y controlar los recursos disponibles.

Un conjunto de herramientas soporte es más que una colección de programas distintos. Hay muchos interfaces entre herramientas, no sólo aquéllos en que se transfieren hacia adelante datos primarios (un segmento de datos de una central) sino también los que pasan lateralmente datos secundarios (por ejemplo: la cantidad de almacenamiento que necesita el segmento de datos) a las herramientas de control. Así,

una de las responsabilidades del centro de herramientas soporte es la de asegurar que estas herramientas puedan integrarse en un sistema aparentemente uniforme y "fácil de manejo" para cada uno de sus distintos tipos de usuarios.

Para el control de configuración, cada módulo requiere una identificación única, que lo distinga de versiones anteriores. Además, se necesita mantener información correspondiente al número de nivel del código fuente original y al de la descripción de los datos, así como al número de nivel del mecanismo (herramientas) que produjo el código objeto. Se obtiene esta información, en caso necesario, acoplando el flujo de la información de control (ver figura 2). Las herramientas insertan los números de nivel automáticamente. La identificación única de cada módulo permite recrear exactamente un módulo en el futuro. También permite a la dirección detectar si se ha utilizado el módulo en otras centrales y en qué lugar, pudiendo decidir la corrección temprana del problema en una operación en vez de esperar a futuras apariciones.

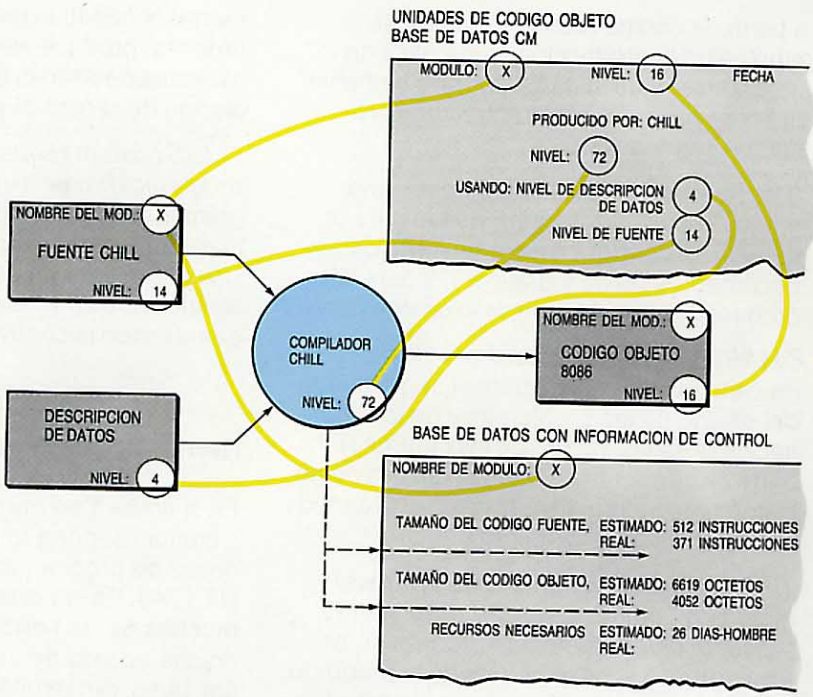


Figura 2
Flujos de información primaria y secundaria en las herramientas.
CM - control de configuración.

Economía de las herramientas soporte

Aunque las herramientas soporte de la programación se consideren frecuentemente como esenciales para el desarrollo y la producción, también se las ve como males necesarios y una fuente de gastos. Sin embargo, tales herramientas proporcionan muchos beneficios tangibles: fiabilidad y mejor calidad del producto final, reducción sustancial en el tiempo necesario para crear el producto, y posibilidad de responder rápidamente a cambios en los requisitos del usuario o a la evolución tecnológica. Debe siempre hacerse un esfuerzo para analizar el coste de las herramientas en función de las ventajas que aportan a la organización.

¿Debería una compañía desarrollar las herramientas que necesita internamente, o bien comprarlas a un suministrador? Si la organización es de tamaño mediano o pequeño, debe, casi con certeza, adquirir todas o la mayor parte de las herramientas. Es poco probable que una pequeña organización pueda abordar la inversión inicial en las facilidades y personas necesarias para su desarrollo. Este exige especializaciones muy diversas, tales como creación de compiladores, diseño de base de datos, y programación de sistemas; ninguna persona, ni siquiera un pequeño grupo, reunirá los conocimientos necesarios.

Incluso una gran organización debería recurrir a suministradores ajenos cuando existan herramientas estándar bien desarrolladas. Las tareas de gestión son relativamente uniformes en una gran variedad de campos de ingeniería y hay muy buenas herramientas de gestión (documentación, control de bibliotecas, sistemas de gestión de base de datos). Para el proyecto ITT 1240 se compraron y adaptaron varios sistemas soporte. Convendría adquirir de antemano los derechos de adaptación o cambio de una herramienta que vaya a utilizarse en un área sometida a evolución. Si estos derechos no pudieran obtenerse, quizá fuera mejor desarrollar la herramienta en la propia organización.

En aplicaciones que requieran herramientas especializadas, puede ser ventajoso encargar su desarrollo a una compañía ajena que tenga la necesaria experiencia. Con ello se evita el gasto de aprender una nueva subtecnología de herramientas, debiendo considerar esto como un medio de obtener algunas herramientas a un coste menor, y quizás con más rapidez. La desventaja de comprar herramientas en el exterior es que su integración no es tan fácil como la de aquéllas que se desarrollan internamente.

Para las organizaciones más grandes, vale la pena hacer una inversión en el desarrollo y construcción de sistemas y herramientas soporte. Un factor importante es que algunas de las herramientas están tan estrechamente relacionadas con el producto final que deben desarrollarse en la misma organización. Como ejemplo, la creación de segmentos de carga de datos es muy característica de la arquitectura del sistema ITT 1240.

Cualquiera que sea el método escogido para obtener las herramientas, el coste de la adquisición de niveles de soporte completos para el desarrollo de un producto será probablemente muy similar. Las organizaciones más grandes pueden permitirse estar en la primera línea tecnológica, manteniendo así la competitividad en sus actuaciones.

Reutilización de las herramientas

Las herramientas más importantes, como las creadas para la central digital ITT 1240, pueden contener más códigos (a menudo más de un millón de líneas de código) que cualquiera de los productos que soportan, y su producción puede costar decenas de millones de dólares durante un número de años apreciable. Esta inversión no puede tomarse a la ligera.

Para maximizar el beneficio de la inversión hecha en el desarrollo de las herramientas soporte, siempre que fuera posible debería generalizarse su estructura de modo que pudieran utilizarse en otros proyectos. Aunque el generalizar aumente el coste inicial del desarrollo en un 20%, este coste adicional se recupera sobradamente en cada nuevo uso de la herramienta. Los ahorros totales pueden ser importantes; varios millones de dólares se ahorraron con la adaptación y nuevo uso de muchos de los componentes del SDSS creado para soporte de las actividades del sistema METACONTA*.

La posibilidad de reutilizar las herramientas depende mucho de la estandarización de los procesos de programación, sus entornos de soporte, y la estrategia general de desarrollo del producto. Hasta el día de hoy, se han conseguido ahorros minimizando el desarrollo de nuevos compiladores, montadores, herramientas de prueba, etc.

La reutilización también se extiende a los ingenieros que manejan las herramientas. El coste de entrenamiento para el uso de un conjunto de herramientas puede ser alto. Si se emplea el mismo conjunto de herramientas en el siguiente proyecto, los costes de reeducación son mínimos, y la productividad será mayor porque los ingenieros ya están familiarizados con ellas.

Tareas de un centro de soporte de herramientas

Un centro de soporte de herramientas proporciona una amplia variedad de servicios. Las diferentes necesidades multinacionales de los centros de ITT exigen que el PSC cree al mismo tiempo interfaces de organización internos y externos para constituir puntos focales para actividades específicas y minimizar así la comunicación entre los diferentes grupos de la organización (Fig. 3). La organización "matricial" interna hace que un director de producto sea responsable de una herramienta o un grupo de ellas, y de atender los requisitos de los usuarios. El resto de la organización proporciona especialistas comprometidos en la planificación, realización, entrega y soporte en el campo de soluciones. También facilita la tarea del desarrollo en múltiples centros, limitando de forma intencionada el alcance de las responsabilidades de los grupos individuales de desarrollo.

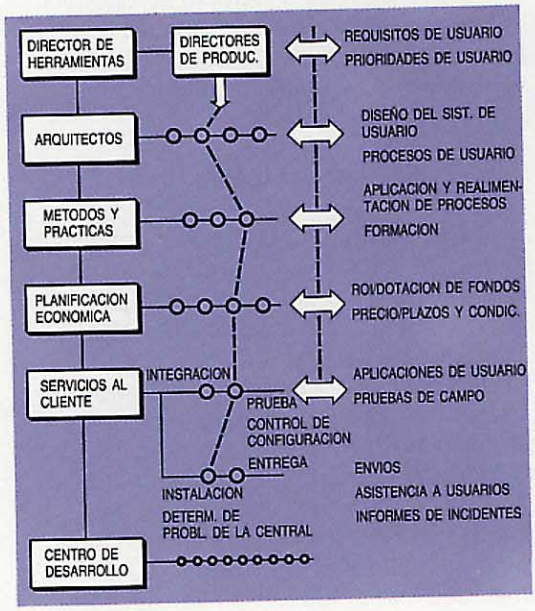


Figura 3
Organización del soporte de programación, mostrando los interfaces con los usuarios.
ROI - beneficio s/inversión.

En la práctica, estas actividades e interfaces no son diferentes de las de otros desarrollos de sistemas y deben cubrir como mínimo:

- requisitos de usuario
- definición de procesos
- arquitectura
- decisión de hacer o comprar herramientas
- desarrollo de cada herramienta
- integración de conjuntos de herramientas
- prueba de calificación
- soporte de distribución e instalación
- funciones de servicio (tratamiento y corrección de faltas y errores).

* Marca registrada del Sistema ITT

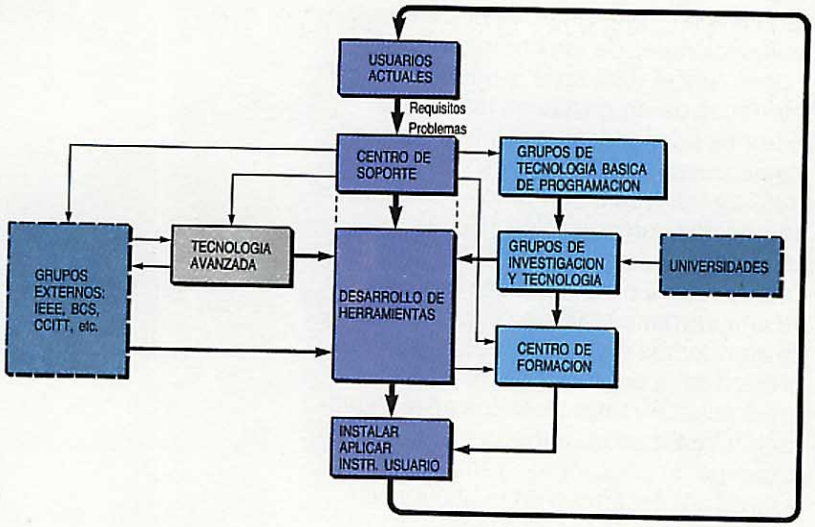


Figura 4
Ciclo de crecimiento de las herramientas soporte de la programación.

Siempre deben considerarse el reconocimiento y asignación de responsabilidades para estas actividades, sea cual fuere la estructura de la organización escogida.

Preparación para el futuro

Un grupo de soporte de herramientas tiene mucho de taller de programación en gran volumen, casi siempre con prisas para no dejar de atender un conjunto inagotable de requisitos de usuario. Sin embargo, es fundamental que el barullo de todos los días no impida la progresión natural de las posibilidades ofrecidas a los usuarios (internos o externos) de las herramientas.

La misión básica final del grupo de soporte es introducir nueva tecnología y mejoras creadas no sólo dentro de las unidades que producen las herramientas, sino también por otros grupos de la organización (si existen) y por organismos externos. En el caso del PSC, las nuevas tecnologías provienen de fuentes internas, tales como los centros de investigación de ITT y grupos de tecnología avanzada, los cuales facilitan una visión nueva y más profunda de los procesos de programación y de los interfaces humanos que el PSC puede transformar en una realidad productiva. El PSC participa en el control y dirección de estas actividades de investigación y desarrollo a través de los grupos de dirección tecnológica de ITT, y además contribuye en sus áreas especializadas de tecnología avanzada. La representación en comités profesionales y de normas (por ejemplo: CCITT, IEEE, BCS, ACM) proporciona entradas secundarias al actual impulso de avance.

También requieren mención especial los lazos que deberían existir entre las herramientas de diseño y fabricación de equipo asistidos por ordenador, que necesitarán reforzarse a medida que las exigencias del sistema y los procesos de diseño se mejoren y se comprendan más profundamente. Esta es un área muy prometedora para la próxima decada, ya que la construcción de un equipo con programas se hace cada vez más difícil y se confía más en la simulación de múltiples niveles en sistemas con ordenadores.

Sin estas actividades encaradas al futuro, el desarrollo de herramientas se fosiliza, los recursos altamente preparados pierden interés y dejan de ser inventivos, y la organización pierde lentamente los beneficios de unas herramientas reutilizables actualizadas.

Conclusiones

El reconocimiento por una organización de la importancia de las herramientas soporte es fundamental para la creación de un grupo con conocimientos especializados de programación, orientado a proporcionar con tiempo herramientas para el desarrollo de nuevos productos. Tal grupo será capaz de adaptar económicamente las herramientas existentes, obviando la necesidad de un nuevo entrenamiento total del usuario, y constituirá un eficaz medio para la introducción práctica de nuevas tecnologías (Fig. 4). Una combinación bien dosificada de desarrollo interno y de programas adquiridos permite optimizar el coste del desarrollo de herramientas, especialmente cuando se une a procesos de estandarización, prácticas y educación extendidos a la organización entera.

Referencia

1 C. G. Denenberg y J. H. Newey: Central digital ITT 1240: Sistemas soporte del producto: *Comunicaciones Eléctricas*, 1981, volumen 56, nº 2/3, págs. 274–282.

Michael P. Saward estudió en la University of London y en el Northampton College of Advanced Technology. Como físico matemático trabajó en la Atomic Energy Authority, Aldermaston, durante tres años; pasó dos años en el Institute of Computer Science, de aquella Universidad, y a continuación entró en los UK Laboratories de IBM. Durante su estancia en IBM, el Sr. Saward dirigió varios proyectos en Europa y Estados Unidos relacionados con productos de sistemas y de programación. Ingresó en ITT en octubre de 1979 como director de programación de STL, y fue nombrado director del PSC cuando se formó en marzo de 1981.

Prevención y eliminación de defectos en programación

Los métodos de prevención y eliminación de defectos consiguen apreciables reducciones en uno de los factores de la programación más costosos y que más tiempo consumen. Se pretende llegar a una programación "cero-defectos" como situación normal.

T. C. Jones
ITT Programming Technology and Development Center,
Estados Unidos de América

Introducción

El elemento más significativo, en coste, de los grandes paquetes de programación, es el de la prevención y eliminación de errores. En los últimos años han surgido dos estrategias sinérgicas para reducir las cotas de error: la prevención de defectos, de modo que la probabilidad de errores sea mínima, y la eliminación de defectos, que detecta y suprime los errores antes de que el sistema pase a ser operacional.

Dada la importancia para la telefonía de unos programas dignos de confianza, ITT estudia intensamente ambas áreas. Realmente una de las primeras actividades esenciales en el desarrollo de la central digital ITT 1240 fue la cuidadosa selección y confección de una serie completa de métodos para eliminar defectos, desde el examen de exigencias e inspecciones de diseño y código, hasta toda clase de pruebas formales. De modo análogo se subrayó la prevención de defectos al elegir los métodos de diseño y especificaciones del sistema ITT 1240.

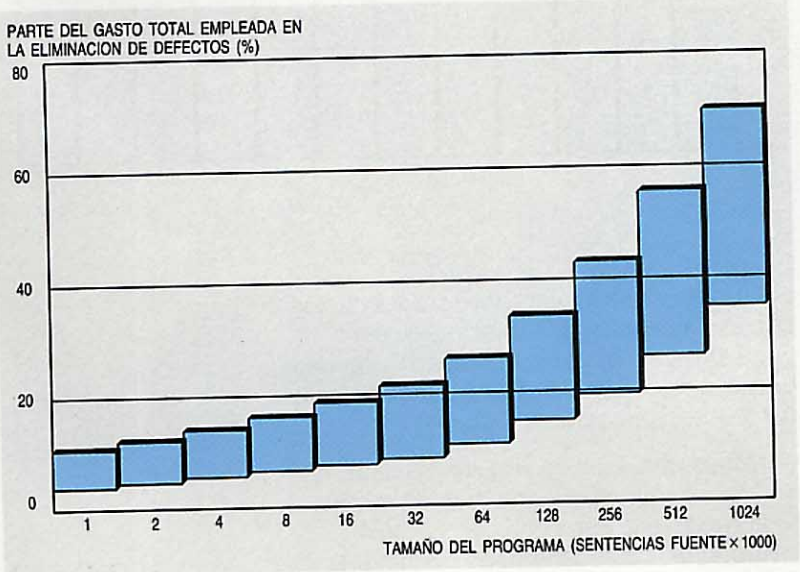


Figura 1
Coste de la eliminación de defectos de programación, mostrando la dispersión para cada tamaño de programa.

El problema de los defectos

La aplicación de la tecnología de programación a las organizaciones comerciales e industriales entra actualmente en su tercera década. Por figurar el diseño y codificación de programas entre los trabajos intelectuales más desafiantes de nuestra era, los errores aparecen más a menudo de lo deseable. Las décadas 60 y 70 vieron un rápido crecimiento, tanto de la programación en sí como de las primeras investigaciones serias en economía y tecnología de programación. Uno de los mayores descubrimientos fue que, para grandes sistemas, la actividad más costosa en todo el ciclo de vida es la eliminación de defectos en los programas[1]. Además, al crecer el tamaño del programa, el coste de tal depuración (revisiones, pruebas, inspecciones, mantenimiento) asciende desde el 10% – 20%, hasta más del 50% del coste total de dicho ciclo de vida (Fig. 1).

Esta verificación condujo rápidamente a buscar las causas de los problemas y su remedio. Un análisis de casi 200 categorías de defectos[2], mostró que la mayoría de ellos procedían de factores relativos a las exigencias, el diseño y las especificaciones de los programas; este hallazgo ha sido la clave para un desarrollo más elaborado de los métodos de prevención de defectos.

Otro importante estudio acerca de las inspecciones del diseño y el código[3] reveló dos hechos esenciales en las nuevas metodologías para eliminar defectos:

- Para detectar defectos, las pruebas valen mucho menos de lo que suele creerse.
- El diseño formal y las inspecciones de código por pequeños equipos de especialistas entrenados, son bastante eficaces en la localización de defectos.

De hecho, la evidencia indicó que el diseño formal y la inspección de código pueden

tener una mayor eficacia total que cualquier otro método corriente de eliminación de errores.

Desarrollos posteriores demuestran que las metodologías de prevención de defectos y de eliminación de defectos están estrechamente ligadas. Uno de estos desarrollos deriva de un estudio acerca del origen de los *módulos propensos-a-error*[4], cuya elevada tasa de errores procede de inadecuadas especificaciones o diseños previos a la codificación. Por falta de especificaciones, algunos módulos no se probaron bien, y no se detectó su condición de propensos-a-error hasta el momento de pasar a operacionales. Al analizar el origen de tales módulos se estableció una conexión directa entre prevención y eliminación de defectos, mostrando, por ejemplo, la medida en que es necesario mejorar las técnicas de diseño cuando se desea incrementar la eficacia de las pruebas.

Otra etapa importante hacia imbricar la prevención y la eliminación de defectos, fue el desarrollo de pruebas de exactitud, las cuales aplican una lógica matemática formal a los algoritmos de programación.

Así, las áreas de prevención y de eliminación de defectos aparecen hoy ligadas. Es evidente que los métodos más eficaces de prevención de defectos son aquéllos que formalizan y clarifican tanto el diseño como la codificación, mientras que para la eliminación son más eficaces los que pueden aislar y eliminar errores, en una y otra de las fases referidas.

Métodos de prevención de defectos

En sentido amplio, la prevención de defectos abarca técnicas que reducen al mínimo los errores humanos, tanto de omisión como de comisión, durante el diseño y codificación de los programas.

Dicha prevención tiene cinco aspectos principales:

- mejora de los métodos descriptivos usados para algoritmos y datos
- reutilización de elementos estructurales de programas existentes y bien construidos
- mejora de los lenguajes de programación utilizados para codificar
- mejora de las técnicas de codificación estructurada
- confección de modelos y prototipos antes de la realización.

Aunque la investigación se desarrolla separadamente en estas distintas áreas, los progresos en cada una de ellas tienden a beneficiar a las restantes.

Métodos descriptivos para algoritmos y datos

El lenguaje natural es ineficaz para expresar el diseño de los sistemas de programación, por la enorme cantidad de texto requerido. Las especificaciones de programas pueden contener más de cien palabras por línea de código fuente. Sorprendentemente el número de páginas por mil líneas aumenta con el tamaño del programa (Fig. 2), si bien a partir de 128.000 líneas disminuye, pues, al parecer no es necesario un diseño tan detallista[4].

El lenguaje natural tampoco es eficaz para describir algoritmos y datos, puesto que los programas versan sobre temas para los cuales dicho lenguaje carece de poder expresivo útil (p. ej., concurrencia, recurrencia, flujo de control)[5].

La búsqueda de métodos para la descripción de algoritmos y datos tiene dos objetivos: mejorar la densidad de expresión (disminuyendo la cantidad de información necesaria para describir un programa), y mejorar la claridad de expresión (aumentando la percepción humana de los rasgos más destacados de un sistema de programación).

Figura 2
Número de páginas en las especificaciones de programas en función del tamaño de éstos.

En la literatura actual sobre programación quizás haya unas 150 notaciones (lenguajes) para mejorar la densidad y claridad de expresión. Simplificando, pueden clasificarse en los siguientes 5 grupos:

- Notaciones basadas en signos matemáticos y lógicos, que mejoran de modo apreciable la densidad de expresión (aunque la claridad puede disminuir).

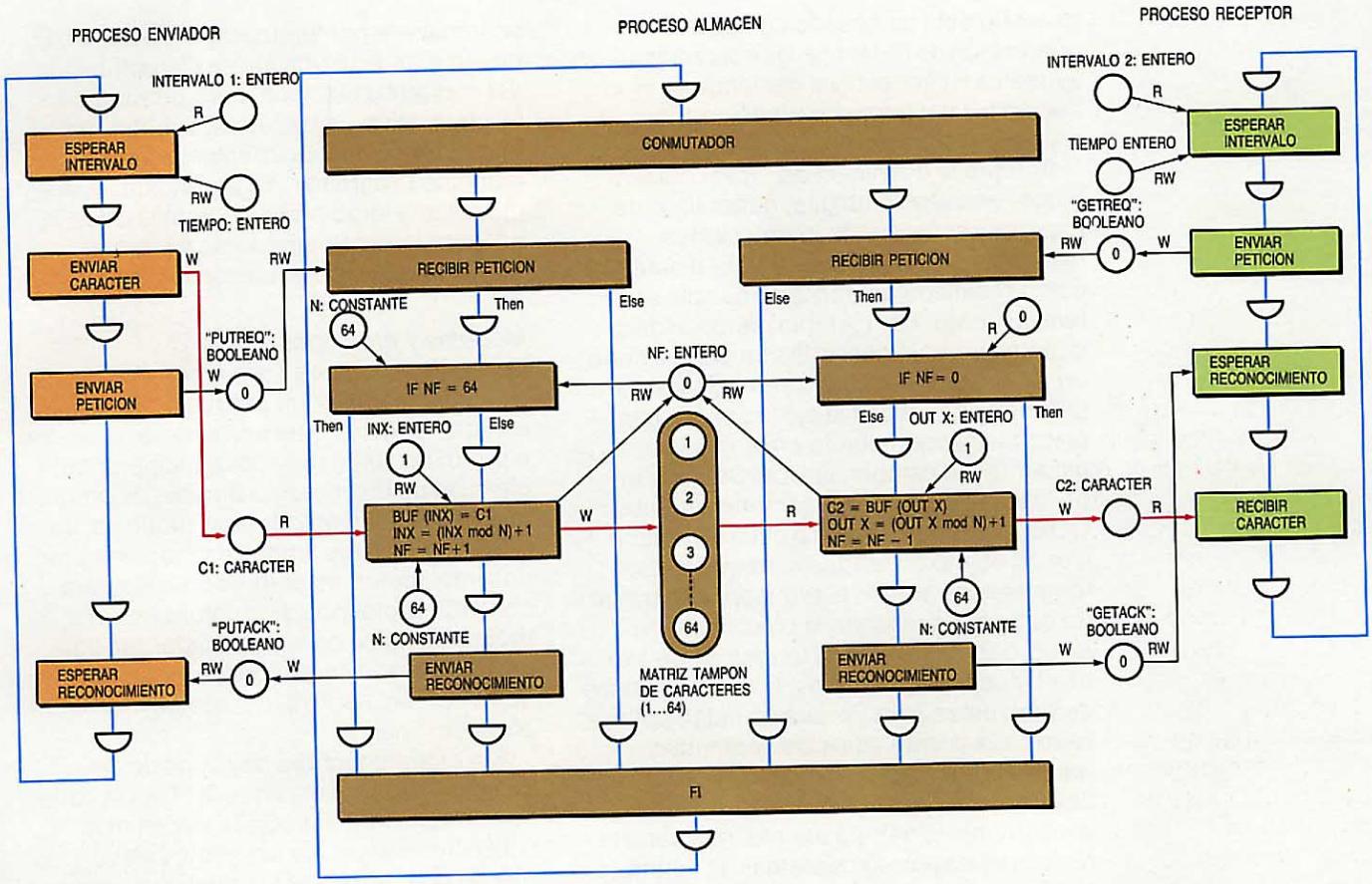


Figura 3
Diagrama Galileo de un proceso enviador-receptor típico.

- Notaciones formadas sobre restricciones sintácticas y semánticas del lenguaje natural (p.ej., inglés estructurado), que mejoran un tanto la densidad.
- Notaciones en varios lenguajes de programación (p.ej., seudocódigos), que mejoran sensiblemente la densidad.
- Notaciones basadas en representaciones gráficas, que dan una claridad mayor.
- Notaciones híbridas, que combinan gráficos, lenguaje natural, seudocódigos y símbolos formales y lógicos, aumentando densidad y claridad.

ITT tiene en progreso planes de investigación en todas ellas. Las conclusiones actuales indican que los métodos más eficaces utilizan: notaciones matemáticas y lógicas para los algoritmos clave, lenguaje natural para discusiones, seudocódigos para diseño inicial de módulos, y notaciones híbridas con gráficos para análisis amplios y debates con los usuarios finales.

El lenguaje de diseño GALILEO desarrollado por el Centro de Investigación de Standard Eléctrica (Madrid, España) es uno de los métodos híbridos de ITT (Fig. 3) que expresa actividades concurrentes en notación gráfica, combinada con notaciones formales para algoritmos clave y lenguaje natural para material explicativo general[6].

Reutilización de elementos estructurales

Desde los primeros tiempos de la programación, funciones corrientes como las rutinas de calendario y tratamiento de fechas, o las pruebas de números primos, se han ubicado en bibliotecas con objeto de volver a utilizarlas en otros programas. Recientemente, la evidencia de que tanto los programas de aplicación como los de tiempo real tienen funciones repetitivas, ha convertido la reutilización en un objetivo importante dentro del desarrollo de la programación. Uno de los métodos de prevención de defectos más eficaces descubiertos hasta ahora, consiste en la creación de catálogos de *funciones certificadas*, que han sido ampliamente validadas y cuya fiabilidad se reconoce.

Un concepto afín es el de catalogar no sólo funciones certificadas, sino también estructuras básicas de rutinas comunes, e incluso programas completos. Así, cuando se necesite un nuevo programa, quizá no sea preciso crear un diseño original, sino simplemente seleccionar del catálogo una estructura existente y cierto número de módulos certificados. Esta técnica es eficaz tanto en prevenir defectos como en la mejora de la productividad[7].

Mejora de los lenguajes de codificación

Aunque se han creado más de 250 lenguajes de desarrollo de programas, sólo

ahora se está calibrando su valor en la prevención de defectos. Dos características ayudan a tal fin: el nivel del lenguaje, y la semántica del lenguaje respecto a los problemas para los cuales se emplea.

Aunque la definición de "nivel del lenguaje" sea algo ambigua, generalmente expresa el número de instrucciones máquina objeto que pueden ser ejecutadas como resultado de escribir una sola sentencia fuente. Por ejemplo, se considera que un lenguaje ensamblador básico tiene un nivel 1, porque cada sentencia fuente en tal lenguaje provoca la ejecución de una única instrucción objeto en lenguaje máquina. En cambio, el COBOL tiene un nivel 3 ó 4, porque una sentencia fuente COBOL conduce a tres o cuatro instrucciones objeto en lenguaje máquina. Los lenguajes de alto nivel reducen la escritura de código y, por tanto, la posibilidad de error. Además, muchos lenguajes de alto nivel modernos apuntan a tipos específicos de problemas para los cuales están optimizados sus vocabularios de sentencias fuente. Por ejemplo, el lenguaje CHILL, desarrollado por el CCITT para sistemas de telecomunicación[8], posee muchas características útiles para los sistemas de conmutación.

La semántica de un lenguaje se emparenta lógicamente con el tema de la reutilización, ya que las características específicas de un lenguaje de alto nivel (que lo hacen idóneo para un cierto tipo de problema) son la incorporación de funciones de aplicación dentro del propio lenguaje. De este modo, el valor general de los lenguajes de alto nivel en la prevención de defectos, es el de proporcionar elementos funcionales certificados y bien construidos.

Mejora de la estructura del código

En su famosa carta de 1968 a la Association For Computing Machinery[9], el Profesor Edsger Dijkstra declaraba que el uso de saltos frecuentes para transferir control entre rutinas alejadas tendía a crear problemas, ya que elevaba la complejidad combinatoria de ramas ejecutables en un grado tal que hacía casi imposible una prueba exhaustiva, no pudiendo el propio autor del programa prever las complejas interacciones eventuales. Esta carta provocó una reacción inmediata y profunda en el mundo de la programación y se inició una tendencia hacia la *programación estructurada*, la cual exige escribir los programas de tal forma que su ejecución discurra linealmente desde el principio hasta el fin. Esto difiere conceptualmente de los programas tipo *GO TO*, en los que puede haber gran separación entre subrutinas que atiendan a una función común y el control se transfiere por bifurcaciones o saltos. El efecto global de una mejor estructuración del código ha sido sutil pero profundo. La programación estructurada supone realmente un cambio de mentalidad sobre el modo de programar. Ya aprendida, puede modificar a largo plazo los estilos que adopten los programadores y mejorar notablemente la prevención de defectos.

Modelos y prototipos

En muchos campos de la ingeniería, sería insólito desarrollar un producto nuevo sin construir previamente una serie de modelos o prototipos para descubrir y superar deficiencias de diseño. Inicialmente, la programación no solía utilizar tales modelos, ya que los primitivos lenguajes máquina y los ensambladores no eran adecuados para construir prototipos. Por entonces, un programa típico en ensamblador era bastante avanzado para servir de modelo, pues quizá contenía el 75% del código fuente del producto final.

Sin embargo, ahora hay tipos de lenguajes y herramientas que facilitan la construcción de modelos. Cada vez es más corriente ensayar los algoritmos clave y las rutinas principales de los nuevos sistemas en estructuras prototipo, antes de empezar el desarrollo real. También hay *lenguajes intérpretes*, APL, BASIC y FORTH entre otros, que ejecutan inmediatamente una orden o sentencia fuente, sin ensamblar ni compilar.

Una nueva tecnología, similar en concepto pero distinta por naturaleza, es la de generación de programas a alta velocidad. Una descripción del problema sirve como entrada para los programas de generación, consistiendo por ejemplo en una serie de preguntas y respuestas entre el programador y el generador. Completada dicha descripción, el generador asume la responsabilidad y produce el programa. Esta técnica supone un avance hacia la producción directa de aplicaciones por el usuario final.

Métodos de eliminación de defectos

Pruebas

Las pruebas son el medio más usual, aunque no el más eficaz, para eliminar defectos. Durante las pruebas, se ejecuta el código del programa con datos artificiales a fin de ejercitar tanto los caminos lógicos del programa como los límites de sus algoritmos. El problema fundamental en las pruebas es su complejidad combinatoria: casi todo programa de tamaño significativo y que utilice datos, puede ejecutar un número enorme de caminos y variables

cambiantes, e incluso ordenadores de gran velocidad interna de operación consumirían años enteros en probar totalmente un programa de moderada magnitud. Se han desarrollado métodos prácticos que minimizan el efecto de la complejidad combinatoria[10], concentrándose en márgenes limitados por los valores máximos y mínimos de las variables usadas. Otro método es el de las pruebas de campo, en el cual los programas actúan en condiciones reales, con datos vivos.

En gran número de casos, la mente humana sigue siendo la herramienta más eficaz para eliminar defectos (se exceptúan los que implican tiempo, velocidad o manipulación de muchas variables simultáneas). Dos tipos de eliminación de defectos confiados a la capacidad humana son las pruebas de exactitud y las inspecciones de diseño y código. Los métodos de eliminación anteriores a las pruebas en sí suelen denominarse *análisis estáticos*[11].

Para programas de medio y gran tamaño, sin embargo, las pruebas solas no bastan; realmente, la eficacia de probar tales sistemas, medida en defectos detectados, puede ser inferior al 50%[12]. Ya que las pruebas por sí mismas nunca garantizan una alta fiabilidad, la práctica moderna utiliza una combinación sinérgica de revisiones, inspecciones, prototipos y pruebas; esta colección, cada vez más numerosa, de técnicas de eliminación de defectos se denomina *análisis dinámico* y emplea la ejecución controlada de programas para identificar defectos[13].

Pruebas de exactitud

Las pruebas de exactitud se emplean sobre todo para validar algoritmos. Categorías más amplias de defectos, tales como la omisión de un requisito fundamental o la introducción inadvertida de un problema nuevo al corregir un error de código, caen fuera del presente ámbito de estas pruebas. Otra limitación es la propia dificultad de su realización. Puede suceder que la prueba sea en sí incorrecta (no es un caso insólito en pruebas matemáticas), y a menudo se consume en la prueba más esfuerzo que en desarrollar el algoritmo inicial. A pesar de ello, se utilizan cada vez más las pruebas de exactitud para los algoritmos clave. Han demostrado el rigor suficiente para que interese incorporarlas a la eliminación de defectos[14, 15].

Inspecciones de diseño y código

Desde los comienzos de la programación han dado buen resultado diferentes tipos de revisión de diseño, revisión por otra persona, recorrido estructurado de la codificación, y otros análogos. Todos se caracterizan por aplicar la perspicacia y la pericia humana a la detección de errores en el diseño y codificación de programas. Tienen en común la revisión por seres humanos de los productos de un proyecto de programación. Difieren en el rigor de los informes de error, en la mecánica de revisión y en el número de éstas.

Son aspectos importantes de los procesos de inspección[3]: la formación de personal en métodos de inspección, el uso de moderadores habituados a mantener a los participantes concentrados sobre su tarea, disminuyendo roces entre ellos, y los análisis e informes de faltas (manteniendo un completo archivo de los defectos para medir la eficacia de los métodos de eliminación como base de futuras mejoras).

Uno de los aspectos más valiosos de las inspecciones es el de que otros diseñadores y programadores distintos puedan analizar los orígenes de los defectos y así evitar ellos mismos problemas similares. Aunque las inspecciones se consideren como métodos para eliminar defectos, también representan uno de los más eficaces medios de prevención ideados hasta el momento[16].

Eficacia de la eliminación de defectos

Uno de los hallazgos sorprendentes de finales de los 60 fue descubrir que las pruebas sobre grandes sistemas eran a menudo ineficaces. A veces, un ciclo de pruebas completo (prueba de módulo, prueba de integración y prueba del sistema) detectaba menos del 50% de las faltas. El análisis de este fenómeno reveló cierto número de razones de la baja eficacia de las pruebas, así como algunas estrategias adecuadas para mejorarla.

Ante todo se comprobó que el diseño y la especificación de muchos programas están incompletos. Como los casos de prueba suelen derivarse de las especificaciones de programas, las funciones no especificadas pueden quedar sin probarse. También se descubrió que las faltas de programación tienden a acumularse en las secciones más difíciles de los programas, denominadas *'módulos propensos-a-error.* Como éstos son identificables por procedimientos varios (por los diseñadores y programadores, por inspección del código fuente, observación de resúmenes en compilaciones o ensamblajes, y análisis de la estructura, datos y aspectos funcionales de los módulos[17, 18]), ha surgido una estrategia de prueba eficaz, mucho más rigurosa que lo usual, para los posibles módulos propensos-a-error (Fig. 4). Han aparecido además otras técnicas que permiten mejores definiciones de los datos de prueba, y la generación o "cría" automática

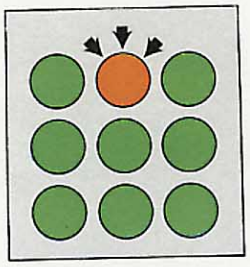


Figura 4
Los módulos propensos-a-error se prueban exhaustivamente, muy por encima del nivel normal.

Figura 5
Las técnicas de prevención y eliminación de defectos se orientan a la programación cero-defectos.

de casos de prueba. Resultado general de estas mejoras es que la eficacia en eliminación de defectos pueda ya superar al 95%, no estimando imposible llegar al 99%.

Conclusiones

Las tecnologías sinérgicas de prevención y eliminación de defectos consiguen avances apreciables en reducir lo que ha sido el elemento más caro de la programación.

Los métodos de prevención de defectos están dirigidos a disminuir los errores humanos, perfeccionando la manera de expresar los conceptos de programación de forma a la vez más clara y compacta que lo hace el lenguaje natural, reutilizando elementos de programas que estén bien formados y sean fiables, y reduciendo la cantidad y complejidad del código que deba crearse para construir nuevos programas. Los métodos de eliminación de defectos se encaminan a detectar y corregir faltas, sobre todo en grandes programas, mediante pruebas, verificaciones de exactitud, e inspecciones formales de código y diseño. Con tecnología de vanguardia, la eficacia en eliminación de defectos puede exceder del 95%, y si se conjuga con los nuevos métodos de prevención podría alcanzarse la meta de una programación cero-defectos.

Los planes de ITT para prevención y eliminación de defectos (Fig. 5) seguirán contribuyendo esencialmente a la fiabilidad de sus sistemas de conmutación para telecomunicaciones, entre ellos la central digital ITT 1240.

Referencias

1 T. C. Jones: Programming Defect Removal: *Proceedings of GUIDE 40, the 40th Meeting of Guide International,* mayo 1975, Houston, Guide International Corp., 1975.

2 A. Endres: An Analysis of Errors and Their Causes in System Programs: *Institute of Electrical and Electronics Engineers Transactions on Software Engineering*, volumen SE-1, nº 2, junio 1975, págs. 140–149.

3 M. E. Fagan: Design and Code Inspections to Reduce Errors in Program Development: *IBM Systems Journal*, 1976, volumen 15, nº 3, págs. 182–211.

4 T. C. Jones: Error Prone Modules Tend to Exist in Large Systems: *IBM Technical Report TRO2.764; Program Quality and Programmer Productivity*: IBM Corporation, General Products Division, 5600 Cottle Road, San José, California, 1977, págs. 6–7.

5 B. Curtis (editor): Human Factors in Software Development: Los Angeles, Institute of Electrical and Electronics Engineers Computer Society Press, 1981, V, 641 págs.

6 F. Vidondo, I. López y J. J. Girod: Galileo: Método para el diseño de sistemas: *Comunicaciones Eléctricas*, 1980, volumen 55, nº 4, págs. 364–371.

7 M. Cavaliere: Hartford Reusable Code a Plus for Productivity: *ITT Commnet*, septiembre 1982, volumen 2, nº 1, págs. 1–4.

8 Especificaciones CHILL: *Comisión de estudio XI del CCITT, foro de realizadores de lenguajes de alto nivel*, noviembre 1979.

9 E. W. Dijkstra: Go To Statements Considered Harmful: *Communications of the Association for Computing Machinery,* marzo 1968, volumen 11, nº 3, págs. 147–148.

10 G. Myers: The Art of Software Testing: John Wiley and Sons, Nueva York, 1979, XI, 177 págs.

11 R. H. Dunn y R. S. Ullmann: Garantía de calidad para programas de ordenador: *Comunicaciones Eléctricas*, 1983, volumen 57, nº 4, págs. 301–306 (en este número).

12 T. C. Jones: Measuring Programming Quality and Productivity: *IBM Systems Journal*, 1978, volumen 17, nº 1, págs. 39–63.

13 E. Miller y W. E. Howden: Tutorial, Software Testing and Validation Techniques, 2a. edición: Institute of Electrical and Electronics Engineers Computer Society Press, 1981, VIII, 499 págs.

14 C. A. R. Hoare: Proof of Correctness of Data Representation: *en* D. Gries (editor): Programming Methodology, recopilación de artículos por miembros del IFIP WG2.3; Springer-Verlag, Nueva York, 1978, págs. 269–281.

15 C. B. Jones: Proofs in Program Development: *en* C. B. Jones: Software Development, A Rigorous Approach, Prentice Hall, Englewood Cliffs, 1980, págs. 74–114.

16 D. P. Freedman y C. M. Weinberg: Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects and Products, 3a. edición: Boston, Little Brown, 1982, IX, 450 págs.

17 M. Halstead: Elements of Software Science: Elsevier, Nueva York, 1977, XIV, 127 págs.

18 T. J. McCabe: A complexity Measure: *Institute of Electrical and Electronics Engineers Transactions on Software Engineering*, diciembre 1976, volumen SE-2, nº 4, págs. 308–320.

T. Capers Jones nació en St. Petersburg, Florida, y estudió en la Universidad de aquel Estado, donde obtuvo un BA en Inglés. Permaneció durante 12 años en IBM, ocupando puestos de dirección e investigación y trabajando en calidad de programas y eliminación de defectos en programación. Actualmente dirige el análisis de tecnologías de programación en el ITT Programming Technology and Development Center. Sus responsabilidades incluyen la evaluación del impacto sobre calidad y productividad de las metodologías de programación y la introducción de técnicas avanzadas de programación en todo el ámbito de ITT.

Garantía de calidad para programas de ordenador

Las consecuencias de los defectos de programación crecen en importancia a medida que los programas son más artificiosos. Si junto a los diseñadores de programas trabajan grupos de garantía de calidad, la incidencia de tales defectos puede cortarse radicalmente.

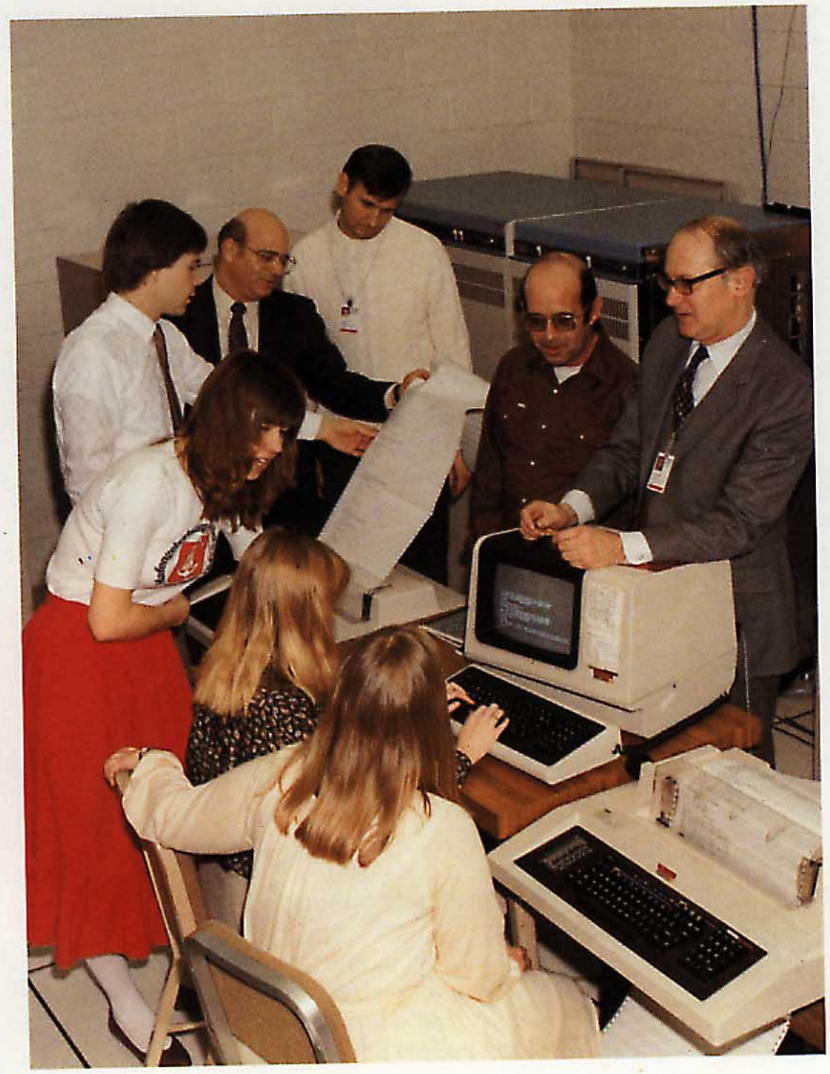
R. H. Dunn
R. S. Ullman
ITT Avionics Division, Clifton, New Jersey, Estados Unidos de América

Introducción

Nuestra sociedad depende en un grado creciente de programas de ordenador. Durante un tiempo, los negocios se han apoyado en sistemas de información programada; actualmente se están adaptando a oficinas electrónicas gobernadas por ordenador. Los sistemas militares y espaciales han estado dirigidos por programas durante muchos años; ahora también las centrales telefónicas y los sistemas de transporte son programables. Incluso los consumidores se ponen en contacto directo con la programación, como por ejemplo en sistemas bancarios automáticos y ordenadores personales.

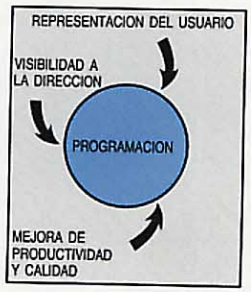
La proliferación de programas conlleva un aumento de coste y el creciente riesgo para las empresas de desarrollar programas que no se terminen a tiempo, o no funcionen como se esperaba. Se han adoptado en la pasada década varias medidas para controlar el desarrollo y mantenimiento de programas, entre las que cabe destacar la creación de grupos de garantía de calidad de programación (GCP). Estos grupos actúan con independencia de los de desarrollo para mejorar la fiabilidad y mantenibilidad de los programas, incrementar la productividad de los programadores y ayudar a la dirección en el control de las actividades de programación. Los ingenieros de calidad asesoran al personal de desarrollo y mantenimiento en la prevención de defectos y eliminación de fallos no evitados, asegurando que no circulen versiones no autorizadas de documentación y código, y analizando la eficacia de los métodos de programación. Dentro de ITT, los grupos de GCP dan soporte a la mayoría de los grandes proyectos de desarrollo de programas destinados a sistemas de instrumentación y conmutación, de los que es destacado ejemplo la central digital ITT 1240.

Los grupos de garantía de calidad dan apoyo independiente a los programadores.



Papel trivalente de la GCP

Un programa acertado de GCP tiene tres misiones primarias: dar a la dirección medios eficaces para observar y controlar el estado del proyecto, representar los intereses de los usuarios de programas, y mejorar la productividad y calidad del producto.



Papel trivalente de la GCP.

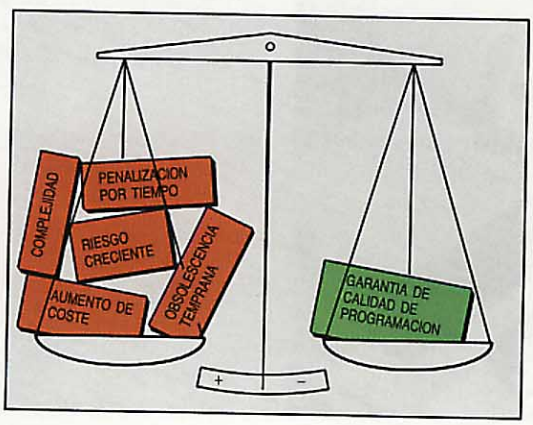
Herramienta de dirección

Es claro que los diseñadores de programas que salen defectuosos, no intentan sino que su producto sea satisfactorio. Por supuesto, muchos proyectos se terminan con éxito, a tiempo, dentro de presupuesto y con pocos defectos. Sin embargo, la dirección no puede dar por sentado que los proyectos de desarrollo de programas concluirán satisfactoriamente, ya que demasiadas veces aparecen grandes dificultades. Más aún, los resultados se desvían de las expectativas tan lentamente, que la dirección no suele enterarse de ello hasta que es demasiado tarde para poner remedios eficaces.

La dirección ha aprendido a preguntar por el estado del proyecto referido a consecución de etapas: por ejemplo, diseño de alto nivel, diseño detallado, producción de código y pruebas. Sin embargo, estos datos son de poco valor si el trabajo que se da como "completado" no se ha sujetado a normas que den un significado real a las etapas. No ayuda el saber que finalizó el diseño de programación de un multiplexor de datos, si ese diseño no puede tratar el 50% de la carga de datos esperada.

El personal de garantía de calidad presta el mismo tipo de servicio para programación que para desarrollo de equipos. Los mecanismos de control asumidos en la función de garantía de calidad, proporcionan la independencia y objetividad necesarias para evaluar cada fase del desarrollo, facilitar a la dirección informes de situación que califiquen el comportamiento real en base a los requisitos establecidos, y atraer la atención del personal de programación hacia posibles problemas en las fases iniciales.

Es común, sobre todo en programación integrada con circuitos complejos, que la dirección imponga una estricta planificación para su desarrollo. En una desesperada tentativa para suministrar el producto a tiempo, hay jefes que no respetan sus preceptos directivos, haciendo que los programadores emprendan tareas antes de haber realizado adecuadamente el necesario trabajo preliminar. En estas circunstancias, la estructura final de la programación acusa forzosamente una base pobre. Un plan eficaz de GCP puede asegurar que no se abandonen las prácticas de desarrollo de una programación sana.



La GCP es importante para la producción de programas eficaces, fiables y de fácil mantenimiento.

Representación de la comunidad de usuarios

Aparte de desear una programación que sea operacional, el usuario busca la calidad. Aunque haya muchos aspectos técnicos en la calidad de programación, la mayoría de ellos están englobados en la definición de calidad dada por ITT, como la capacidad de un producto o servicio de satisfacer a sus usuarios y cumplir sus especificaciones. Comprobando que los diseñadores de programas utilizan métodos correctos, y revisando junto con ellos su trabajo al finalizar etapas, los grupos de GCP ayudan a asegurar que los resultados de cada etapa (documentación o código) se contrasten con los primitivos requisitos del usuario. Las desviaciones encontradas en estas revisiones se corrigen sin más dilación.

Los grupos de GCP participan, a su vez, en la planificación de las pruebas y otras actividades de eliminación de defectos, a fin de comprobar el ajuste de los programas a la aplicación del usuario. Otros modos de representar los intereses del usuario son la promoción de técnicas y disciplinas para elevar la mantenibilidad de la programación, y, cuando sea posible, el análisis del comportamiento del sistema ya instalado.

Mejoras en calidad y productividad

Por estar las actividades de GCP generalmente orientadas a proyectos, el servicio más importante prestado por estos grupos se relaciona con la acumulación de experiencia, y el aprovechamiento de ésta en proyectos presentes y futuros. Para que sean útiles, se deben cuantificar los historiales de los proyectos, llevando estadísticas de todo tipo de defecto, junto con detalles completos de su causa y modo de detección.

Históricamente, los diseñadores se han esforzado poco en medir su propio comportamiento. Por el contrario, los grupos de garantía de calidad han insistido en mediciones del producto, análisis y reprocesos.

Programa de calidad

Una vez definido el papel de la GCP, se puede ahora ver cómo actúa durante el desarrollo de programación. Debe reconocerse que los objetivos de la GCP se consi-

guen mediante la disciplina de trabajar con la calidad inherente a la programación, que no sea resultado de la inspección y corrección del producto final. El alcanzar este objetivo compete esencialmente al personal de desarrollo. Es necesario, por tanto, examinar la GCP dentro de un vasto plan de calidad, que incluya también la comunidad de desarrollo de programación. La sinopsis siguiente se extrae de una definición más completa, referida a una calidad global de la programación[1].

Prevención de defectos

La mayor contribución de la GCP a prevenir defectos es el asegurar que el desarrollo proceda sistemáticamente. Se han definido varias metodologías[2,3] para obtener una descomposición escalonada del diseño y análisis de problemas, bases para un desarrollo controlado. Este queda garantizado por medio de inspecciones, revisiones, y auditorías al final de las etapas que resulten naturalmente de la metodología. Los ingenieros de calidad de programación insisten en establecer normas para la consecución de cada paso, comprobando el grado de cumplimiento de esas normas, y asegurando que se han aplicado con prontitud las medidas correctoras oportunas. Cuando transcurren meses o años entre la aprobación de la especificación de requisitos y el comienzo de las pruebas, hay un gran riesgo de que aparezcan desviaciones de las normas y por lo tanto se requiere el más estricto control. La figura 1 presenta, para una cierta metodología de diseño, resultados tangibles en varias de las actividades intermedias que el grupo de GCP puede evaluar durante esta etapa.

Debe reconocerse que, por muy estrechamente que se controle un proyecto, está condenado al fracaso si su planificación es incorrecta. Según esto, los ingenieros de calidad revisan la factibilidad de las etapas previstas y la dotación de personal. También examinan las técnicas y herramientas destinadas al análisis y diseño, y revisan las pautas planeadas para codificación y prueba. Así mismo,dichos expertos ayudan al personal de desarrollo en la planificación de pruebas que demuestren el cumplimiento de las especificaciones.

El creciente uso de los recursos del ordenador (memoria, espacio en disco, tiempo de ejecución, etc.) es un aviso anticipado de que el proyecto puede encontrar problemas susceptibles de originar defectos. Los ingenieros de calidad insisten en tener estimaciones tempranas de los medios que se requieran, y en que dichas estimaciones se actualicen en las etapas apropiadas. Las tendencias de crecimiento

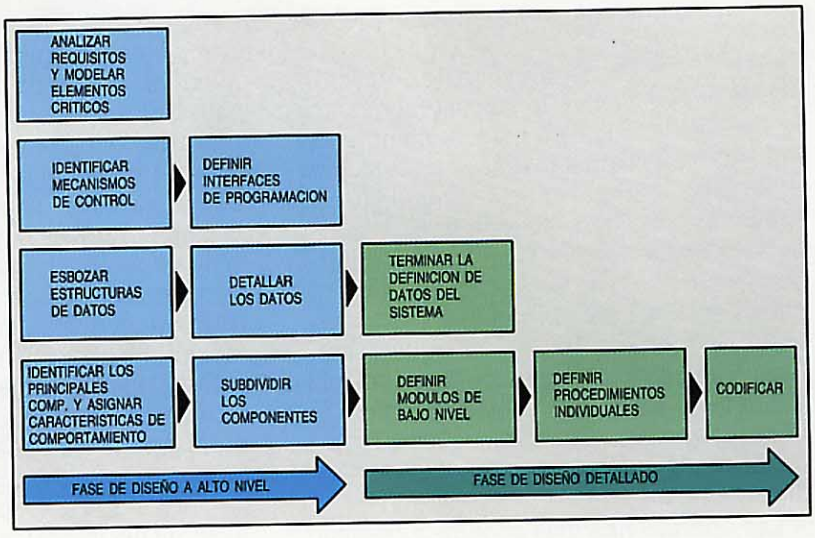


Figura 1
Proceso de descomposición del diseño de programación.

se analizan para determinar si amenazan la factibilidad del proyecto, y si proceden en cualquier momento acciones correctoras (desde alterar los niveles del personal hasta redefinir los objetivos técnicos).

El código ya utilizado previamente y que se ha probado en uso real, puede esperarse que tenga menos errores que el código nuevo. Sin embargo, la inclusión de "módulos reutilizables" en nuevos sistemas de programación, está rodeada de varios problemas. No obstante, en la medida posible, los ingenieros de calidad certifican las características de comportamiento de tales módulos, de forma que puedan ser incorporados con toda confianza en nuevos programas.

Merece destacarse el que gran parte del análisis de datos de fallos, se dirige a la prevención de defectos futuros.

Detección de defectos

La prevención de defectos puede reducir el número de errores latentes, pero no eliminarlos totalmente en ninguna etapa. Hay que admitir que se producirán defectos en las fases de requisitos, diseño y codificación. Es notorio que todas ellas, excepto la de requisitos, se inician con los resultados de la fase precedente. Así, se necesita una política capaz de detectar estos defectos en la primera oportunidad posible, antes de que se hayan propagado a las fases posteriores y exija por tanto un coste mucho mayor su eliminación.

La primera ocasión de eliminar un defecto de un documento de requisitos o diseño, es inmediatamente después de haberlo introducido. Por consiguiente, la filosofía de GCP aboga por una inmediata revisión de toda la documentación, actuando como se describe en otro artículo de esta publicación[4]. Los grupos de GCP no sólo aseguran que las revisiones se hacen en el momento

Seis medios para asegurar la calidad de programas de ordenador.

oportuno, sino que participan activamente en ellas.

La eliminación de defectos de codificación puede dividirse en dos categorías: estática y dinámica. La primera comprende procedimientos que no requieren ejecución del código, mientras que la segunda hace referencia a pruebas activas. Las técnicas estáticas exponen directamente los defectos, mientras que las pruebas dinámicas provocan fallos por los cuales se localizan defectos latentes. Así, los métodos estáticos son más eficaces en horas de trabajo por defecto eliminado, si bien se siguen necesitando las pruebas dinámicas, pues no todos los defectos son detectados por tales métodos.

Las operaciones más comunes de eliminación estática son las de inspección de código[4] y de análisis estático. Este último denomina colectivamente a una serie de técnicas relacionadas, que emplean herramientas de programación para descubrir fallos semánticos y de estructura y revelar propiedades inadvertidas. El análisis estático reduce el esfuerzo requerido para inspección de código. Otros métodos estáticos incluyen la ejecución simbólica, o seudoejecución de entradas algébricas, y la prueba de exactitud, tediosa y propensa a errores (y por ello raramente usada), en la que el código se reduce a un esquema de proposiciones demostrable en un sistema formal de lógica. Todos estos métodos estáticos de exposición de defectos se tocan ligeramente en la referencia 1, y se detallan más extensamente en una publicación de la IEEE[5]. Aunque tales métodos son eficaces, aún quedan defectos a ser detectados por pruebas dinámicas. Los grupos de GCP contemplan estas pruebas como series de actividades: prueba unitaria de cada elemento del programa, integración de éstos en un conjunto, y prueba de calificación. Dependiendo de la aplicación, pueden estar involucradas otras secuencias de prueba: integración de los programas individuales en un sistema distribuido, integración de los programas del producto en el equipo que van a controlar, pruebas de instalación, etc. Incluso las secuencias básicas están sujetas a variación. Por ejemplo, la identidad – aunque no la función – de las pruebas unitarias como actividad separada, puede perderse en una metodología de integración por niveles.

De cualquier modo que se realice, la división de las pruebas dinámicas en segmentos únicos proporciona un medio de validar el código en cada etapa de dichas pruebas, frente a las etapas de desarrollo correspondientes, permitiendo así confiar en la pureza de la estructura de programación. Esto se refleja en la figura 2, tomando los programas de un producto como ejemplo. Las pruebas unitarias demuestran que cada módulo del sistema satisface los requisitos establecidos; las pruebas de integración validan el diseño; la integración equipo/programas confirma que el sistema cumple las especificaciones de la programación; y la calificación del sistema prueba formalmente que el diseño se ajusta a las exigencias del usuario. Para asegurar que se cumplan estos objetivos, los grupos de GCP revisan toda la documentación previa a las pruebas (tal vez incluso preparando los procedimientos de prueba para la calificación) y supervisan la forma en que se realizan tales pruebas, garantizando su adecuación a los procedimientos establecidos.

Un enfoque consiste en diseñar las pruebas individuales conociendo totalmente la estructura de programación y con relativa indiferencia a las especificaciones del usuario; otro consiste en olvidar la estructura y atender por entero a las especificaciones. En la revisión de los planes y procedimientos de prueba, los grupos de GCP buscan asegurar que se cubren ambos enfoques, así como los intermedios, para tener certeza de que no se escape ninguna oportunidad de observar un defecto. También insisten en que las pruebas se diseñen para ejercitar los programas en condiciones de máxima tensión, por ejemplo al máximo de aviones en un sistema de control de tráfico aéreo.

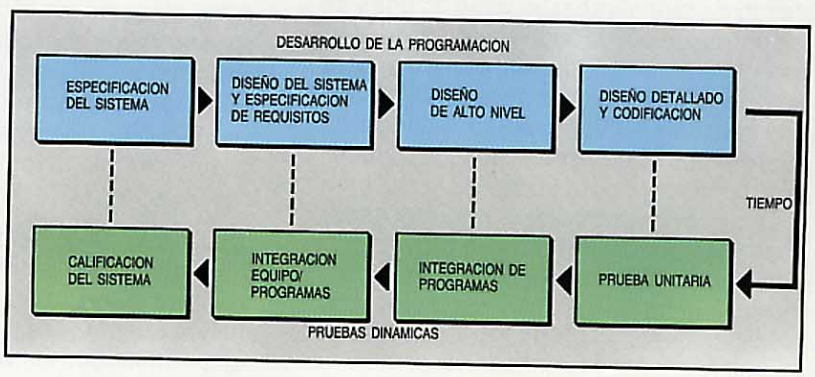


Figura 2
Validación del código segmentado.

Control de configuración

Pocos productos se pueden modificar tan fácilmente como los programas de ordenador y su documentación. Se denomina *control de biblioteca* al mecanismo de control de cambios en programas, en sus distintas formas (inicialmente como especificaciones de requisito, finalmente como especificaciones funcionales, código, y quizá microprogramas). El control de la biblioteca de programas se puede encajar dentro de la función de GCP, o bien ésta puede auditar el control que otros realicen.

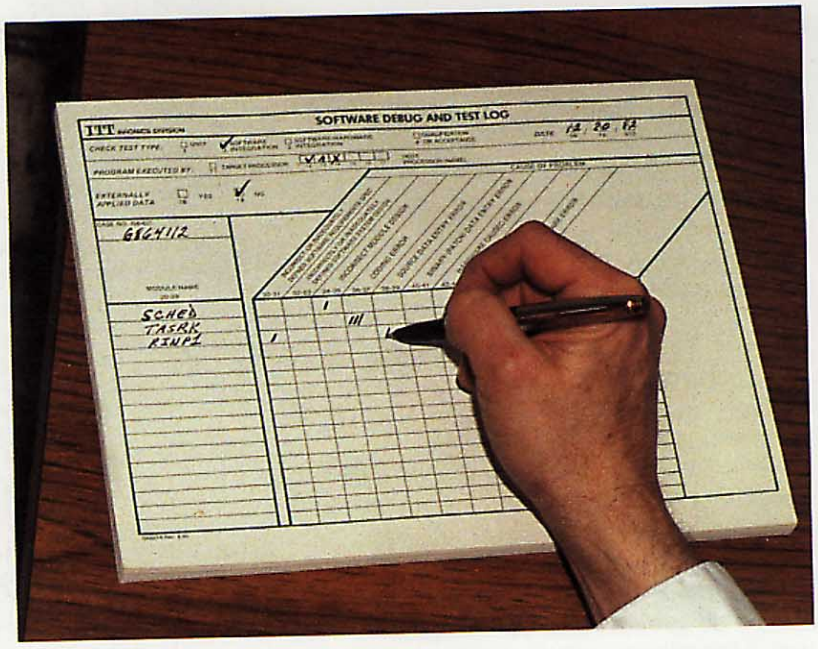
En varias etapas del desarrollo se definen las bases de estabilización de documentos y del código. Por ejemplo, justo después de la revisión de una especificación de requisitos, o después de la prueba satisfactoria del código contenido en una "construcción" de los elementos de programa que llevan a cabo una función determinada. Posteriormente, cualquier cambio de código o documentación sólo puede hacerse siguiendo procedimientos controlados. Estas prevenciones disipan cualquier duda sobre qué versión de programación se está probando o suministrando al usuario, y aseguran la coherencia de documentación y código indispensable para el mantenimiento de la programación.

Análisis de defectos

El ingrediente final del programa de garantía de calidad probablemente sea el de efectos más duraderos. Antes se han expuesto las medidas realizadas por los grupos de GCP, y el papel que juegan en la mejora de calidad y productividad. Específicamente, estos grupos tratan de hallar dónde se han introducido los errores y la eficacia de las técnicas de detección de defectos.

La fotografía muestra una forma de registrar datos de defectos. El formato se rellena diariamente para cada proyecto en el que exista actividad de prueba. La mayoría de los espacios del formato se usan para anotar el número de defectos producidos por las diversas fuentes, los cuales se diagnosticaron después de aparecer el fallo. La primera línea identifica el tipo de prueba; la segunda y tercera, el entorno de la prueba. Dada esta información, es posible medir la eficacia de las técnicas de detección de defectos. Por ejemplo, si se encuentran más errores de código imputables a diseño detallado en integración que en prueba unitaria, ello indica que las técnicas de prueba unitaria no son las idóneas.

Mucho se puede aprender de la identificación de las fuentes de defecto, unida al modo de detectar dicho defecto. Normalmente se podría esperar que un sistema tuviera más errores en codificación que en los requisitos. Supóngase, sin embargo, que hay el mismo número de cada tipo. Esto sugiere, bien que el personal sea muy experto en la escritura del código, o bien (más razonablemente) que deberían mejorarse las técnicas y herramientas para la definición de requisitos. Durante largos años, los técnicos de garantía de calidad han utilizado análisis de este tipo, donde se buscan eventos u objetos que resalten del resto (a menudo llamados análisis de Pareto), como medio para mejorar la calidad y la productividad.

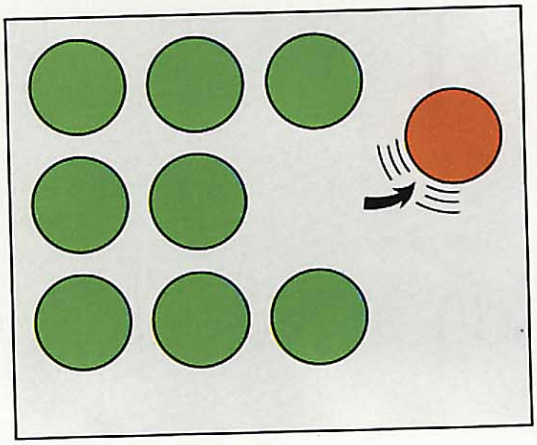


Cuaderno de depuración y prueba de programas.

Usos inmediatos del análisis de defectos

El análisis de defectos puede también tener un valor inmediato. La mayoría de los grandes sistemas de programación parecen tener al menos un módulo que ocasiona un número de defectos excesivo. Por cualquier razón — por ejemplo, una especificación de requisitos escasamente definida o la mala elección de un programador — la estructura de estos módulos hace que por cada defecto eliminado haya riesgo de introducir uno nuevo. Lo mejor que se puede hacer con un módulo así es rediseñarlo desde el principio. El análisis de Pareto (distribución de defectos por módulo), puede servir para detectar la existencia de módulos propensos al fallo.

Los datos de defectos tienen otro uso inmediato: suponiendo una correlación entre defectos y fallos (imprecisa, pero adecuada a nuestros fines), se pueden hacer análisis de tendencias y así estimar el tiempo de prueba requerido para reducir el nivel de fallos del producto lo suficiente como para poder utilizarlo. Entre las técnicas de análisis más interesantes,



Los módulos de programación con un excesivo nivel de defectos, deberían ser rediseñados desde el principio.

figuran las que usan los modelos de fiabilidad de programación[1,6] para deducir distribuciones de fallos con respecto a horas o a tiempo de ejecución.

Garantía de calidad de programación dentro de ITT

En ninguna industria conocen estas actividades una expansión mayor que en el Sistema ITT. Desde 1976, un Consejo de Calidad de Programación ha coordinado los esfuerzos de GCP en Europa, donde ITT ocupa en tales funciones a más de 60 ingenieros, repartidos entre nueve unidades. Un conjunto de 13 normas de calidad de programación, comunes a las unidades europeas, cubren las actividades de GCP en auditorías, revisiones, inspecciones, control de configuración, pruebas y fiabilidad. Específicamente se dirigen a sistemas de conmutación, incluída la central digital ITT 1240.

Inspirada inicialmente en los requisitos del Departamento de Defensa de EE.UU., ahora existe GCP en muchas compañías ITT de Norteamérica dedicadas a telecomunicación y fabricación electrónica. Más aún, el Consejo de Calidad de Programación de ITT North America cuenta entre sus miembros a representantes de ITT en el campo de seguros y productos de consumo. Ocho unidades de ITT en EE.UU. desarrollan planes de GCP. Pese a la diversidad de aplicaciones de programación, en todas las unidades se han seguido los preceptos comunes de GCP; el Consejo sirve también como vehículo para transvasar experiencias entre las diferentes compañías. Como puede suponerse, se mantiene una estrecha relación entre el Consejo Europeo y el Americano.

Conclusiones

Como se describe en otro lugar de esta publicación, los productos ITT dependen cada vez más de la programación; de ello es un claro exponente la central digital ITT 1240. Aunque el origen de la garantía de calidad no se remonte al inicio de los desarrollos de programación, durante los últimos años se ha planificado eficazmente la GCP en muchas unidades. Esta tendencia se reforzará a medida que ITT tenga que apoyarse más en la programación.

Referencias

1 R. Dunn y R. Ullman: Quality Assurance for Computer Software: Nueva York, McGraw Hill, 1982, VIII, 351 págs.

2 G. D. Bergland y R. D. Gordon (editores): Tutorial: Software Design Strategies: Institute of Electrical and Electronics Engineers Computer Society, 1979, VII, 415 págs.

3 G. D. Bergland: A Guided Tour of Program Design Methodologies: *Computer,* octubre 1981, volumen 14, nº 10, págs. 13–37.

4 T. C. Jones: Prevención y eliminación de defectos en programación: *Comunicaciones Eléctricas,* 1983, volumen 57, nº 4, págs. 295–300 (en este número).

5 E. Miller y W. E. Howden (editores): Tutorial: Software Testing and Validation Techniques: Long Beach, Institute of Electrical and Electronics Engineers Computer Society, 1978, III, 425 págs.

6 R. Dunn: Estimating the Reliability of the Software Black Box: *ITT Commnet Focus on Technology,* diciembre 1981, volumen 1, nº 2, págs. 1–4.

Robert H. Dunn se graduó BS en ingeniería física y MS en matemáticas. Se incorporó a ITT en 1957 como ingeniero de desarrollo. Comenzando con un sistema en tiempo real de gran escala, desde 1965 ha ocupado varios puestos de gestión de programas de ordenador en ITT Avionics. En el presente, el Sr. Dunn es asesor de calidad y fiabilidad de programas de ordenador, y presidente del Consejo de Calidad de Programación de ITT North America. También ha sido portavoz de la industria en varios comités gubernamentales.

Richard S. Ullman se graduó como ingeniero mecánico por la Universidad de Nueva York en 1956, y en 1966 obtuvo el título master por el City College de aquella ciudad. Después de trabajar para Kollsman Instrument Corp. y Ford Instrument Co., el Sr. Ullman se incorporó a ITT Avionics en 1971 como director de ingeniería de calidad. Actualmente es vicepresidente y director de la división, con responsabilidad sobre la calidad de todos los sistemas militares y comerciales. Es presidente del comité de fiabilidad y mantenibilidad de la National Security Industrial Association y miembro activo del Institute of Electrical and Electronics Engineers, de la American Society for Quality Control, y de la Society of Logistics Engineers.

La tecnología de sistemas de coordinación como base para un entorno de programación

Un examen detenido de los problemas de coordinación de las actividades es parte esencial en el desarrollo de un entorno completo de programación. La utilización de la teoría rol/actividad y los sistemas de coordinación representan aspectos importantes del trabajo emprendido por ITT.

A. W. Holt
H. R. Ramsey
J. D. Grimes
ITT Programming Technology and Development Center, Stratford, Connecticut, Estados Unidos de América

Introducción

Ningún problema ha sido tan molesto para la industria de los ordenadores ni para los usuarios de sus productos como el de crear programas que conviertan un ordenador de aplicación general en herramienta útil para un determinado fin. A pesar del amplio esfuerzo de investigación y desarrollo realizado durante décadas por gobiernos, industrias y universidades, nadie está satisfecho aún con los resultados. En general, el desarrollo y producción de programación continúa siendo costoso y poco fiable.

La programación presenta, sin duda, especiales problemas para organizaciones como ITT, ya que muchos de sus productos básicos (sobre todo en la telecomunicación) dependen de complejos programas de aplicación concreta, que deben ser sumamente fiables para resultar útiles. Considérese, por ejemplo, la central digital ITT 1240, basada en una red de microprocesadores distribuidos que operan independientemente. Ya en lo técnico la tarea es muy ardua, al comprender una compleja colección de programas concurrentes, pero a nivel organizativo es aún más difícil, por participar centenares de programadores en varios países, con complejas relaciones formales e informales.

En investigación y desarrollo se aborda este problema por caminos diversos, que incluyen lenguajes de programación, métodos para mejorar la reutilización de los programas y componentes afines, herramientas para la rápida construcción de prototipos, y la llamada "programación automática", basada tal vez en el concepto de inteligencia artificial. Uno de los enfoques más nuevos, en el que trabajan intensamente muchos sectores de la comunidad, es la creación de entornos* de programación.

Entorno de programación

Un entorno de programación, en este contexto, es una estructura completa, diseñada al efecto, que permite a los programadores, a sus jefes, y al personal auxiliar, cumplir con eficacia la función de programación. En cierto sentido, han existido entornos de programación desde que se comenzó a programar. Sin embargo, los componentes lógicos de estos primitivos entornos no se diseñaron a la vez, ni tampoco con el fin principal de prestar ayuda a la actividad de programación. Lo nuevo, entonces, es el enfoque consciente del diseño sobre el entorno total de programación.

Diseñar un entorno de programación significa preparar la integración de múltiples herramientas, tareas diferentes, personas distintas. La práctica ha ido advirtiendo gradualmente a los especialistas que muchas de las dificultadas reales del desarrollo de sistemas complejos, se relacionan, de una forma u otra, con problemas de integración. Los grandes proyectos de programación – como todas las grandes empresas industriales – no son obra de un solo programador, sino de un grupo de programadores en colaboración. Además, el éxito

* Un *entorno* ** es un conjunto organizado de *herramientas* y de *medios* que prestan soporte a una *organización de personas especializadas* que realizan algún tipo de *trabajo*.

** Varios términos utilizados en este artículo tienen un significado técnico propio en la teoría rol/actividad. En general, son ya familiares al lector (*rol, acción, interacción, estado*) y su sentido técnico se acerca bastante al que tienen en el contexto de las organizaciones interpersonales, aunque la correspondencia no sea siempre exacta. Se ha utilizado la letra itálica para resaltar su uso como términos técnicos.

depende no sólo de la eficiencia y fiabilidad con que se realice la parte técnica de la programación, sino también de la calidad de interacción de los programadores entre sí, con sus jefes, con el personal auxiliar, con los suministradores y con los usuarios de sus productos.

ITT se ha comprometido en un importante esfuerzo de investigación y desarrollo en el campo de los entornos de programación, cuya justificación se basa en lo siguiente:

- Aunque la investigación y desarrollo sobre entornos de programación sea relativamente nueva, ofrece un futuro prometedor.
- Los beneficios potenciales del diseño de entornos aumentarán apreciablemente en el futuro próximo, a medida que se disponga de mejores estaciones de trabajo y equipos de trazado gráfico.
- No existen entornos de programación ya preparados que ITT pueda adquirir e instalar para satisfacer sus necesidades.
- La tecnología básica requerida para construir entornos de programación coincide con la que se precisa para informatizar otras clases de trabajos. De este modo, la investigación sobre dichos entornos puede contribuir a las capacidades de ITT en otras áreas, como la automatización de oficinas y fábricas.
- ITT tiene los recursos técnicos necesarios para emprender tal programa de investigación.

Proyecto de entorno de programación

Este artículo presenta uno de los proyectos de ITT en el tema de los entornos de programación, basado en nuevos conceptos del dominio de la teoría de sistemas y de la programación de sistemas.

La principal diferencia entre este proyecto y otros desarrollos de entornos actuales, es que se centra en elevar la productividad de los grupos, por encima y más allá de lo que pueda conseguirse proporcionando mejores herramientas a los individuos. Específicamente este proyecto introduce nuevas técnicas informáticas para coordinar las actividades de muchos colaboradores en un proyecto único. El resultado es un gran incremento en:

- la integración de los productos del trabajo
- la integración de las herramientas y de los medios que las producen
- el soporte directo al modo de trabajo escogido por la organización de desarrollo (p. ej., procedimientos, métodos, y estándares)
- el soporte directo a los cambios que se produzcan en la organización del proyecto a lo largo del tiempo.

Además, dichas técnicas tienen efectos secundarios beneficiosos. Primero, mejoran también la productividad individual porque el individuo, al realizar una tarea compleja, debe "coordinarse consigo mismo". Segundo, son relevantes para la coordinación de muchos módulos de programas o de equipo. En otras palabras, pueden considerarse como nuevas técnicas para organizar y controlar el procesamiento concurrente. También afectan a una gama de cualidades deseables, como son la capacidad de recuperación y de reorganización, la fiabilidad y la reutilización del código almacenado. Naturalmente, desde el punto de vista del entorno de programación, estos efectos son de gran importancia, y no meramente "secundarios".

El enfoque de ITT

El enfoque de ITT presenta dos novedades básicas. La primera es la utilización de un nuevo concepto teórico – llamado *teoría rol/actividad* – para describir las actividades humanas, organizadas de manera útil a la construcción de un soporte por ordenador. Esta teoría debe considerarse como un estudio del comportamiento de la organización, aunque su origen no esté en la sociología. Más bien procede de la investigación informatizada sobre temas tales como la comunicación, concurrencia y cooperación entre entidades interdependientes, ya sean personas o microprocesadores.

La segunda característica original es la utilización de un nuevo concepto, denominado sistema de coordinación, como fundamento de los entornos de programación. Todas las demás construcciones de entornos conocidas por los autores se basan en un sistema operativo convencional y en una base de datos corriente de complejidad variable. El concepto de sistema de coordinación con el que trabaja ITT, implica un conjunto de primitivas arquitectónicas (rol, acción, interacción) fundamentalmente diferente al de los sistemas operativos clásicos (fichero, programas, estructura de directorio, proceso). Estas primitivas se derivan de la teoría rol/actividad.

Un tercer aspecto importante del proyecto – no tan original – es la construcción de herramientas y medios para apoyar las tareas de programación *per se*. La construcción del sistema de coordinación representa sólo una pequeña parte de la formación del entorno de programación. Gran parte del trabajo está en la construcción de las herra-

mientas y medios, variados y de niveles múltiples, necesarios para dar apoyo sustancial a los complejos proyectos de programación. El comportamiento de tales herramientas viene, desde luego, afectado en gran manera por su residencia en un sistema de coordinación genuino. Además, el modo de partición y de diseño de las mismas habrá de reflejar esta diferencia.

Ciertas herramientas o medios se necesitarán universalmente, en todos los proyectos de programación, mientras que otras se clasificarán por tipo de proyectos, o por

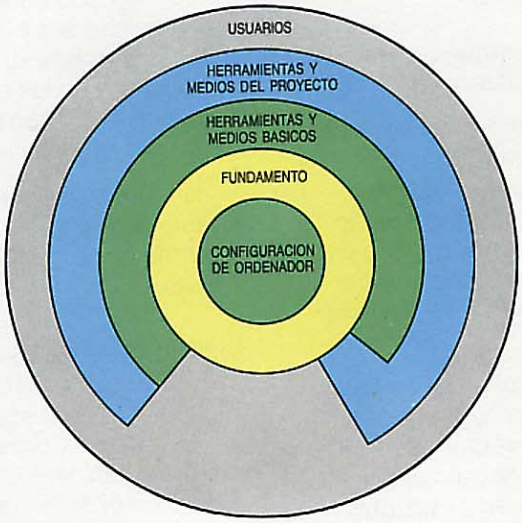


Figura 1
Estructura general del entorno.

proyectos individuales. A la primera categoría pertenecen las destinadas a:

- mantenimiento del entorno de programación
- desarrollo de herramientas y medios
- "ayuda" y "enseñanza/aprendizaje", en forma genérica
- mantenimiento de la biblioteca y de la base de datos, archivado, comunicaciones generalizadas (correo, por ejemplo), preparación conjunta de documentos, control de la configuración, etc., todo de manera genérica
- simulación y análisis de estructuras rol/actividad
- edición de textos y gráficos.

Específicas de diversas clases de proyectos, o de proyectos individuales, serán las herramientas y medios que incorporen particulares disciplinas de dirección de proyectos o de programación, lenguajes, o estilos. Estas se consideran como "envolventes" de las herramientas y medios básicos, según ilustra la figura 1.

En cualquier caso, lo crucial del problema es el sistema de coordinación y el nuevo comportamiento que lo acompaña, derivado como antes se indicó de la teoría rol/actividad.

Teoría rol/actividad

El correcto funcionamiento de toda organización humana depende de la moral, del buen juicio y de unas reglas. El ordenador no da un apoyo directo en lo que concierne a la moral o al buen juicio, pero puede ser de enorme ayuda en todos los aspectos del trabajo condicionados a reglas. Por lo tanto, si se van a utilizar ordenadores para prestar eficaz soporte al trabajo organizado, es necesario aprender a reconocer qué aspectos de dicho trabajo se sujetan ya, o pueden sin perjuicio sujetarse, a unas reglas. También hay que aprender a describir, analizar y diseñar tales reglas. En cualquier caso, debe advertirse que el ordenador sólo puede prestar ayuda si existen unas reglas o normas.

En general, las reglas que gobiernan el trabajo organizado son numerosas, y se interrelacionan de maneras complicadas, muchas de ellas poco conocidas o admitidas por la gente. Se sujetan a reglas temas tales como:

- "qué" va en "qué" cajón
- cuándo es "demasiado tarde", y qué hacer al respecto
- a quién "dárselo" o "decírselo", y de qué modo
- cómo "manejarlo" o cómo "completarlo"
- dónde "consultar".

Por supuesto, también se incluyen las cosas que primero se ocurren al pensar en reglas, como es el organigrama, las normas de seguridad, normas de reembolso, directrices, procedimientos, estándares, etc.

La teoría rol/actividad se ocupa de las redes de reglas que gobiernan el trabajo. Ofrece conceptos básicos útiles para considerar, y, empíricamente, investigar dichas redes de reglas; también proporciona notaciones para describirlas de forma que resalten sus propiedades importantes. Los términos de la teoría están claramente relacionados con los requisitos de aplicación de la regla.

Los dos conceptos más importantes de esta teoría son el *rol* y la *interacción*. A través de ellos es posible investigar, configurar, registrar, analizar y diseñar estructuras que expresen los aspectos sometidos a reglas de trabajos en los que cooperen muchas personas. Puesto que son parte de una teoría matemática, estos conceptos tienen propiedades, y en consecuencia

definiciones, matemáticas. No obstante, lo que aquí nos preocupa es la manera en que estas definiciones se aplican al mundo real. Como ocurre siempre con los términos técnico-científicos, la aplicación buscada es aproximadamente coherente con su significado ordinario, aunque más limitada.

El contenido e implicaciones de los roles e interacciones no son fáciles de entender con rapidez. Sólo pueden comprenderse enteramente sus consecuencias a través de entrenamiento y de trabajo práctico.

Rol

La lista siguiente señala los aspectos más importantes del término *rol:*

- Siempre existe en la organización una *responsabilidad* asociada a cada rol.
- Los roles – cuando están activos – son asumidos por personas que detentan la responsabilidad asociada. Distintas personas pueden asumir un rol particular, pero solamente una puede hacerlo en un momento dado. Por otro lado, una persona puede asumir roles diferentes, posiblemente a la vez.
- Asociados con cada rol hay materiales que intervienen críticamente en la realización del rol y que pueden clasificarse en partes tales como herramientas, trabajo en curso, marcas y etiquetas de estado especial, etc.; al conjunto se le denomina *cuerpo del rol.* Estos cuerpos de rol siempre ocupan espacio; cada uno pertenece a un rol, y sólo a uno, al mismo nivel de descripción.
- El cuerpo del rol tiene en cada momento un *estado* perteneciente a una serie potencialmente grande de estados posibles. Dicho estado cambia de acuerdo con ciertas reglas y como resultado de la realización del rol, denominándose también *estado del rol.*
- Se llama *acción* a lo que ocurre entre un estado y el siguiente. Hay un conjunto de acciones posibles asociado a cada rol; una acción ocupa siempre un tiempo.
- Con cada rol existe un conjunto de *comportamientos* posibles, cada uno de los cuales puede caracterizarse como secuencia de estados/acciones. Las reglas que gobiernan el comportamiento del rol expresan, de forma cerrada, el conjunto de posibles secuencias de estados/acciones.

Interacción

Los principales aspectos de una interacción son:

- A cada interacción se asocia una lista de roles, denominada *conjunto participante* de la interacción. Siempre que se produce una interacción, intervienen en ella *N* actores (uno por cada rol del conjunto participante), cada uno de los cuales realiza una acción concreta, la misma en cada ocasión. Un ejemplo sencillo de interacción es el ingreso de fondos. El conjunto participante es el cliente y el cajero del banco; la acción del cliente es la de entregar el dinero, y la del cajero recibirlo. Las dos acciones se combinan para formar un evento único.
- En consecuencia, dentro de una interacción debe existir un pre-estado y un post-estado determinados para cada participante. La interacción comienza cuando todos los participantes se encuentran en los pre-estados adecuados, y finaliza cuando todos ellos han alcanzado sus respectivos post-estados.
- No todas las acciones de un rol tienen que ser parte de una interacción con otros roles; pueden ser *acciones solitarias.*

Aplicaciones de los conceptos

Estos conceptos técnicos pueden servir para analizar una actividad de un trabajo. Como ejemplo, consideremos el control de la configuración.

El control de la configuración se considera normalmente como una *responsabilidad,* y por tanto es natural contemplarlo como un rol. (Obsérvese que no tiene por qué ser asignado a una sola persona; su realización puede resultar de un conjunto de responsabilidades, ninguna de las cuales se ocupe exclusivamente del control de la configuración.) Quizá no exista un responsable de la configuración en la organización; incluso no se mencione el término control de la configuración en ninguna descripción de funciones. No obstante, el rol existe, y aquí se le da el nombre de *responsable de la configuración.* Esta misión puede ser asignada a un individuo o puede circular libremente entre un conjunto cualificado de personas.

El paso siguiente consiste en comprender la misión del responsable de la configuración. Para hacerlo, es necesario considerar de qué acciones depende de modo inmediato el que asume el rol, y qué acciones dependen más directamente de él: en otras palabras, con qué otros roles participa en las interacciones. Varios roles posibles acuden a la mente: *probadores de módulos,* el *definidor de la configuración,* y el *cliente de la configuración.* Como interacciones básicas tenemos: la transferencia del módulo desde el probador hacia el responsable de la configuración, la transferencia de la definición de la configuración desde el definidor hacia dicho responsable, y la

transferencia de una copia de la configuración desde este último al cliente.

Muchas interacciones, aunque no todas, implican que se transfiere algo (X) de un rol a otro rol. En tales casos, la razón es que los derechos y/o la responsabilidad sobre *X* pasa de un rol a otro rol, lo cual explica la transferencia de un módulo desde el probador al responsable de la configuración. Por otro lado, cuando dicho responsable entrega una copia, no cede la responsabilidad sobre la configuración sino sólo sobre esa copia, mientras que el receptor adquiere el derecho a utilizarla en el desarrollo de su función.

Existen diversas maneras de llevar a cabo una transferencia; una de las más corrientes es desplazar un objeto físico de un lugar a otro. Cuando el objeto a transferir es información almacenada en forma electrónica, los métodos utilizados no son necesariamente sencillos. Ciertamente la operación "transferir esta información desde un cuerpo de rol a otro" no está entre las posibilidades estándar de los ordenadores existentes. En cambio, se dispone de órdenes diversas de lectura y de escritura, y de controles de derecho de acceso. El propósito de las transferencias antes mencionadas suele realizarse con ayuda de tales órdenes. Donde éstas resulten insuficientes, habrán de complementarse con acuerdos entre los usuarios sobre quién realizará y quién no podrá realizar diversas acciones en ciertos momentos.

En la teoría de rol/actividad, todos los eventos inter-rol son interacciones, y cada interacción implica la *sincronización* de todos los participantes en la misma. Todos ellos deben estar preparados para intervenir en la interacción antes de que empiece, y todos deben salir de ella a la vez, con un conjunto de acuerdos equiparado acerca de lo que ocurrió, si la interacción procedió conforme a las reglas. La teoría mantiene que esta forma de comportamiento es indispensable para las actividades humanas organizadas. Sin embargo, la utilización normal de los ordenadores no presta un soporte directo a la sincronización de múltiples participantes en el sentido que acaba de explicarse. Esto significa que la coordinación de operaciones con soporte de ordenador exige la adhesión de los usuarios a reglas de temporizaciones y de comunicación mutua no implantadas en ordenadores, si desean que sus relaciones de mutua dependencia lleguen a feliz término.

Un sistema concebido para control de configuración, descrito en términos rol/actividad, da medios para evaluar una implantación real, así como para preguntar hasta qué punto el sistema encaja en la organización de trabajo de que forma parte (p. ej., si se puede adaptar adecuadamente a los cambios previsibles, si permite aplicar el sentido común cuando fallan las reglas, etc.). Finalmente, el análisis rol/actividad de un sistema de control de configuración (o de otras tareas), puede descubrir nuevas y mejores formas de utilizar los ordenadores en dichas actividades.

Ideas sobre los sistemas de coordinación

La práctica actual en entornos de trabajo informatizados consiste en construirlos sobre un sistema operativo unido a una base de datos, más o menos potente, que se considera como el centro del entorno en su conjunto. Al sistema operativo se le encomienda el permitir operaciones concurrentes sin que los usuarios se interfieran entre sí perjudicialmente. A la base de datos se le pide que sea el conector universal entre las personas y entre las herramientas. Puesto que el acceso común no restrictivo crea caos en la organización, se utilizan mecanismos de control de acceso relativamente estáticos. Estos mecanismos, por su propia naturaleza, son más adecuados para la definición de cuerpos de rol que para la implantación de transferencias en el sentido ya tratado.

El *sistema de coordinación* * es un nuevo concepto sobre el cual fundamentar un entorno. Este concepto se aplica adecuadamente hoy en forma de programas de sistema, pero puede en el futuro acercarse a la lógica de control del equipo implicado. Desde el punto de vista actual, un sistema de coordinación es análogo a un sistema operativo, e incluye muchas de las estructuras que normalmente formarían parte de la base de datos central asociada.

Como se indicó antes, los sistemas de coordinación se contemplan en ITT como una posible base apropiada para entornos de programación. Consideremos el punto de vista del usuario sobre su actividad dentro de tal sistema de coordinación, especialmente en contraste con el contexto de un sistema operativo convencional.

En un sistema operativo, el usuario se ve a sí mismo como alguien que hace que los programas se ejecuten en su provecho, sobre sus propios archivos o conjuntos de datos, o sobre otros a los que tenga acceso. El puede, hasta cierto grado, reflejar la separación de sus roles por medios tales como los directorios múltiples.

* El propósito de un sistema de coordinación es unir entre sí un conjunto de usuarios que trabajan concurrentemente, mediante relaciones de interdependencia que puedan describirse. Estas relaciones deben ser conceptualizadas en términos de rol/actividad y describirse en la notación denominada *diagramas de rol/actividad.*

Normalmente, el ordenador está realizando una secuencia de tareas para el usuario. Aunque éste pueda entender que muchas de estas tareas son *concurrentes* desde el punto de vista *lógico* (es decir, se refieren a asuntos independientes que le afectan, como responder al correo o pasar una compilación), esto no se refleja de forma apreciable en el modo de ejecución de estos programas por el sistema operativo.

En el contexto del sistema de coordinación, el usuario se ve como *actor* de un conjunto de *roles,* en cada uno de los cuales tiene acceso a un cuerpo de información (el cuerpo del rol).

Una vez registrado en el sistema, el usuario puede examinar el estado de sus roles y realizar acciones en cualquier rol donde se precise una acción suya, o ésta sea posible. Algunas acciones serán solitarias, mientras que otras formarán parte de interacciones con roles entonces controlados por otros usuarios.

Un usuario puede automatizar sus acciones cuando ello sea oportuno. Esto quiere decir que puede sustituir su propia actividad en el terminal por un guión preparado ejecutable, llamándose *automáticas* tales acciones; si, en cambio, la acción requiere la presencia del usuario ante el terminal, se denomina *participativa.* La interacción entre dos actores (dos usuarios, cada uno en sus roles respectivos) puede ser enteramente automática, semiautomática y semiparticipativa, o totalmente participativa. En el primer caso ninguno de los usuarios precisa registrarse. En el último caso, ambos deben participar personalmente para que la interacción se complete.

Así, en principio, todos los *roles* de un usuario pueden ser *activos* sin su participación personal. Sólo si la interacción es participativa, el correspondiente rol esperará a la intervención del usuario en una sesión presente o futura con el terminal. Es importante observar que tal rol no se considera interrumpido; toda interacción consume un tiempo, y se supone que el usuario efectuará su contribución antes de que transcurra el tiempo permitido total.

La acción automática es un medio particularmente eficaz para la prestación de servicios a una comunidad de usuarios que trabajan en un sistema de coordinación. Por ejemplo, una determinada persona puede ser designada responsable del servicio de compilación de CHILL en un proyecto de programación. El cuerpo del rol del responsable CHILL contendrá, entre otras cosas, el mejor compilador de CHILL entonces disponible, herramientas para el mantenimiento del compilador y toda clase de registros relativos a las actividades del servicio de compilación que se está supervisando. En circunstancias normales, el uso del compilador por un diseñador de aplicaciones se estructuraría como una interacción (o serie de interacciones) con el sistema de compilación CHILL. La acción por parte del responsable CHILL sería, en circunstancias normales, automática (aceptar la fuente para compilación, quizá comprobar su validez, aplicarle el compilador y devolver el resultado de la compilación). El responsable CHILL podría hacer que las interacciones del servicio fueran menos automáticas, si las circunstancias lo exigieran (p. ej., cuando aparecen preguntas sobre el compilador que no pueden ser contestadas automáticamente). En la misma línea, otra persona podría encargarse de ofrecer servicios de base de datos.

Para ayudar a visualizar el modo de operación, la figura 2 muestra un simple diagrama para la producción de un informe mensual. Esta figura muestra el lenguaje-diagrama rol/actividad, lenguaje gráfico formal para describir el comportamiento de organizaciones. En términos generales, el diagrama muestra dos roles (el responsable del informe y el distribuidor del informe) y una clase de rol, a saber, la de autor, que representa a los autores que producen las secciones del informe mensual.

Las líneas verticales sobre las que se colocan cajas, representan comportamientos del rol. Cada caja corresponde a una acción, y cada segmento de línea vertical entre cajas representa un estado (del cuerpo del rol); cuando las cajas que expresan acciones de dos roles diferentes se unen entre sí por un segmento de línea horizontal, ello indica que ambas acciones son parte de una interacción.

Un diagrama de este tipo, añadiendo más detalles sobre las acciones, puede utilizarse para generar un conjunto de cuerpos de rol

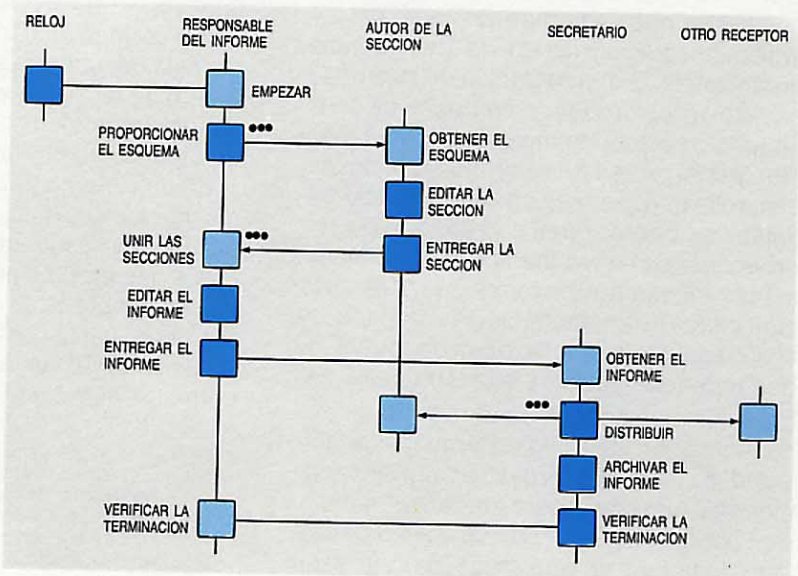


Figura 2
Diagrama rol/actividad para un procedimiento de informe mensual.

adecuados, que a continuación se podrá distribuir entre las personas que asuman esos diversos roles. El comportamiento de tales personas será entonces regulado por el sistema de coordinación, según se indica en el diagrama.

Se han diseñado varios sistemas que leen e interpretan descripciones de comportamientos gobernados por protocolos. Un auténtico sistema de coordinación debe hacer mucho más: de modo análogo a los sistemas operativos, ha de gestionar la ejecución concurrente de un conjunto de cuerpos de rol interconectados (cada cuerpo de rol se asemeja, en cierto grado, a un proceso). El cuerpo de rol debe estar bajo control de la persona que asuma la responsabilidad correspondiente, y su comportamiento debe poderse modificar por esa persona según dicte la necesidad y el sentido común. En un sistema de coordinación bien desarrollado, se realizarán pruebas para asegurar que las modificaciones introducidas por los actores participantes concuerdan con el propósito de los diagramas rol/actividad, tales como el procedimiento de informe mensual mostrado en la figura 2.

Como mínimo, un sistema de coordinación debe aplicar los criterios rol/actividad siguientes:

- cooperación de roles concurrentes, con implicaciones en el control y en la responsabilidad del rol
- comportamientos de rol tanto participativos como automáticos
- control formal de las interacciones como partes de comportamientos, coincidentes con cambios de estado en los cuerpos de los roles participantes.

Se espera que el sistema de coordinación hará un amplio uso de gráficos, generalmente del tipo mostrado en la figura 3. Se ha tomado la fotografía de un prototipo gráfico sencillo utilizado para idear puestos de trabajo electrónicos. Mediante ventanas gráficas superpuestas se presenta al usuario un contexto visible de la tarea que muestre explícitamente las tareas suspendidas. El diálogo habrá de ser profusamente gráfico, y deberá imponer al usuario cargas de procesamiento de información mucho más bajas que las exigidas por los métodos convencionales de diálogo.

Conclusiones

El enfoque utilizado por ITT para el desarrollo de un entorno avanzado de programación implica dos grandes innovaciones: su fundamento en la teoría rol/actividad, y un sistema de coordinación basado en la dicotomía rol/actividad como mecanismo operativo fundamental para las diversas herramientas y medios que el sistema ofrece.

Figura 3
Un prototipo de interfaz gráfico.

Es claro que el enfoque seguido aquí proporciona al usuario una sensación diferente. El modelo conceptual, con el cual el usuario puede entender el comportamiento de su sistema soporte, es mucho más familiar y más orientado a la tarea que los sistemas soporte basados en ordenador del pasado. El diseño de las herramientas y de los medios en sí viene afectado notablemente por la potencia, flexibilidad, y naturalidad de este modelo. Es una forma genuinamente nueva de concebir la utilización de ordenadores por seres humanos, así como los sistemas soporte informatizados. Los autores creen que este enfoque encierra grandes posibilidades, no sólo de aplicación inmediata a los entornos de programación, sino también en otros muchos aspectos de los productos electrónicos de ITT, y en entornos profesionales.

Anatol Holt se graduó BS en matemáticas por Harvard en 1950, MS en matemáticas por el MIT en 1952, y PhD en lingüística descriptiva por la Universidad de Pennsylvania en 1963. Trabajando para Sperry Rand, colaboró en la autoría del primer sistema compilador/macroensamblador mundial para el ordenador Univac I. De 1964 a 1975, dirigió el proyecto de teoría de sistemas de información en ADR/Compass. Después de un periodo como profesor de ciencias informáticas y como director del centro de cálculo de la Universidad de Boston, ingresó en ITT en el grupo de tecnología avanzada de programación del PTDC, donde se dedica al diseño de entornos de programación basados en sus investigaciones previas sobre la teoría de sistemas.

Rudy Ramsey se graduó PhD (Universidad de Oklahoma en 1970) en psicología experimental. Posee una experiencia considerable de investigación y de aplicación en ciencias informáticas, incluyendo el diseño e implantación de lenguajes de programación, sistemas

de traducción-escritura, herramientas de programación y diversos sistemas interactivos. Ha trabajado como psicólogo experimental aeroespacial en la US Navy, como ingeniero senior en Martin Marietta, y como científico senior en Science Applications, Inc. Siendo investigador senior en ITT Programming, el Dr. Ramsey contribuye señaladamente a la investigación sobre sistemas de coordinación, como especialista en factores humanos aplicados a la informática.

Jack Grimes recibió su PhD en ingeniería eléctrica por la Universidad Estatal de Iowa y su MS en psicología experimental por la Universidad de Oregon. Es director de tecnología avanzada de programación en ITT Programming. Este grupo desarrolla arquitecturas de entornos de trabajo informatizados para mejorar la productividad de grupos. Anteriormente, fue director de desarrollos avanzados en ordenadores de mesa en Tektronix.

Entorno de aprendizaje interactivo para enseñanza de la programación

Está en desarrollo un sistema de enseñanza por videodiscos, para producir cursos de programación interactivos, individualizados y adaptados a la velocidad de aprendizaje del estudiante. El sistema permitirá a los programadores estar en vanguardia tecnológica.

H. F. Moody
F. Wilson
ITT Programming Education Center,
Stratford, Connecticut,
Estados Unidos de América

Introducción

En el entorno organizativo actual, basado en los ordenadores, hay una creciente demanda de enseñanza y entrenamiento en programación. El Programming Education Center de ITT se creó para enseñar técnicas de programación a los ingenieros que producen programas para los productos ITT, y técnicas de gestión a quienes dirigen a los programadores. Para ello se están desarrollando cursos y sistemas nuevos para enseñar la programación, que permitirán a ITT mantenerse en vanguardia de la tecnología y control de la programación.

Además el Centro ofrece minicursos a los ejecutivos no relacionados con la programación, para concienciarlos de la creciente importancia de la misma en el éxito de todas las áreas comerciales de ITT. Prácticamente en todos los productos y servicios del futuro, desde los nuevos sistemas de telecomunicación hasta los seguros, la programación jugará un papel fundamental.

Una actividad esencial en la enseñanza de la programación es transferir tecnologías nuevas y consistentes en el seno de ITT. Actualmente se están desarrollando cursos sobre centrales digitales ITT 1240 que puedan ser utilizados por instructores de compañías de ITT dedicadas al desarrollo y la producción de tal sistema.

La creciente demanda de educar en programación ha creado la necesidad de nuevas herramientas. Dos factores limitan la expansión de este tipo de enseñanza: la escasez de instructores cualificados y los altos costes del transporte de los estudiantes a los centros de formación y su alojamiento. Se ha desarrollado una gran variedad de sistemas de autoenseñanza – por medio de diapositivas, películas, cintas de vídeo, instrucción asistida por ordenador, y, por supuesto, la palabra escrita – para intentar superar la falta de instructores y reducir los costes. La mayoría de estos sistemas tratan de proporcionar formación interactiva, individualizada y adaptable al estudiante. Por desgracia, las dos primeras características han sido difíciles de conseguir, ya que actualmente hay pocos sistemas que sean realmente individualizados o interactivos.

Entorno de aprendizaje interactivo

Los recientes desarrollos de sistemas de vídeo y microordenadores han abierto un camino hacia la creación de sistemas de formación interactivos, capaces de responder a las necesidades individuales. En particular, el videodisco permite almacenar miles de imágenes visuales y miles de millones de octetos de datos. Al combinar el videodisco con un ordenador, se obtiene un sistema auto-reconfigurable según los requisitos específicos de cada tarea y capaz de interactuar con dicha tarea a cualquier nivel deseado, lo cual es de suma importancia.

El Programming Education Center de ITT está desarrollando un prototipo de sistema de vídeo interactivo (IVES: Interactive Video Education System), que consiste en dos reproductores de videodisco por láser controlados por un microordenador. La capacidad y diseño del IVES le permiten integrarse en la red ITT. Esto permitirá a todas las compañías ITT disponer de información coherente actualizada y de instrucción sobre herramientas, técnicas y productos. Este sistema puede llegar a formar

Videodisco óptico. Disco de plástico de 305 mm que puede almacenar 54.000 imágenes fijas; el reproductor del videodisco puede visualizar cualquiera de esas imágenes en 5 segundos como máximo. También se pueden almacenar hasta 30 minutos de imagen en movimiento.

parte del puesto de trabajo de los programadores del futuro.

Dispositivos del sistema

Uno de los principales componentes del IVES es el videodisco de láser (ver fotografía). Este dispositivo, con una enorme densidad de almacenamiento, es capaz de contener 54.000 muestras de información por cara y acceder a cualquiera de ellas en menos de cinco segundos. Puede presentar la información de tal modo que simule un instructor que cada vez enseña a un cierto alumno. Así, puede servir de base a un sistema que facilite información a muy alto nivel a un ejecutivo, a un ingeniero información detallada, y a los técnicos procedimientos de reparación, obteniéndolo todo de la misma base de datos. El mismo videodisco puede también utilizarse en un sistema, controlado manualmente, para aplicaciones de marketing (p.ej., proporcionar imágenes para las presentaciones), o como un archivo de información (para el trabajo de los programadores).

Además del propio videodisco los principales componentes son los reproductores, un microordenador con sus periféricos que aporta la inteligencia necesaria para interactuar con los usuarios, un controlador de vídeo universal y un monitor de vídeo en color.

Programas del sistema

Los programas del sistema tienen dos funciones básicas: interactuar con el alumno y controlar el videodisco.

La configuración del prototipo IVES (Fig. 1) será evaluada en diferentes centros ITT, en Europa y Norteamérica. Los resultados permitirán a ITT evaluar y elegir los medios para satisfacer las necesidades de formación en el futuro, así como los dispositivos a utilizar siempre que se precise un acceso interactivo a información almacenada. Es probable que también se descubran nuevos usos del IVES.

Utilización del sistema

El IVES supone el primer paso para llevar al aula la tecnología del futuro y ofrecer a los estudiantes el mejor entorno educacional posible. Las simulaciones interactivas y realistas de los trabajos de los programadores enriquecerán el entorno de aprendizaje a medida que el alumno interactúa con el material del curso. Está comprobado que una persona que trabaja de modo interactivo con un adecuado material de formación, aprende más aprisa y retiene mejor que quien adopta una actitud pasiva.

El principal problema de los sistemas utilizados hoy en la formación es que los dispositivos y programas del sistema en sí pueden ser tan complejos que distraigan la atención del estudiante. Para que éste pueda concentrarse en el material a estudiar, es necesario que el sistema que se utilice como medio de aprendizaje sea lo más transparente posible. El IVES se ha diseñado para producir las mínimas distracciones.

El entorno de aprendizaje IVES puede tomar diversas formas. Con el sistema completo, varios estudiantes pueden seguir el curso al mismo tiempo, lo cual permite tanto la instrucción individual como la de grupo, posibilitando la interacción entre individuos. Otra posibilidad es utilizar los propios elementos del sistema como soporte de una clase convencional, obteniendo de la información almacenada en el disco miles de imágenes para ilustrar y reforzar la enseñanza. Una tercera posibilidad es el empleo de la información para investigar en un campo específico o revisar el material dado en la clase.

Con esta tecnología es posible que un estudiante pueda acceder simultáneamente al saber de varios profesores, superando el clásico esquema de un solo profesor por alumno.

Evaluación del sistema mediante un curso de telefonía

El Programming Education Center (PEC) de ITT está produciendo un curso de telefonía que será evaluado por las compañías de ITT en Norteamérica y Europa. Este curso, que se eligió por la gran audiencia que tiene, proporciona a los ejecutivos, controladores,

programadores e ingenieros lecciones de telefonía individualizadas, desde los orígenes hasta los nuevos sistemas, incluyendo la central digital ITT 1240. El estudiante recorre todos los sistemas de ITT, analizando cada tipo de sistema al nivel de detalle que desee. Se utilizan dos videodiscos: uno contiene miles de imágenes fijas, obtenidas de diferentes compañías europeas y americanas, dentro y fuera de ITT, y el otro contiene 30 minutos de imágenes animadas, junto con voz y música para acompañar al primer vídeo. La interacción humana con el sistema se consigue a través de dispositivos periféricos diseñados para controlar el grado de detalle requerido. En algunas áreas el estudiante puede investigar los detalles a nivel de componente telefónico.

La evolución del IVES responderá a las reacciones de las diferentes compañías en los aspectos de niveles de interacción, adquisición y retención de conocimientos, gestión de archivos del sistema, capacidad de medidas y factores humanos. También se considerarán posibles mejoras, como transmisión de programas por la red y el control del IVES por unidades de ordenador o microordenador.

El curso de telefonía está basado en un viaje a través del tiempo, guiado por unos caracteres gráficos generados por el ordenador. Además, la simulación gráfica de una máquina del tiempo permite al estudiante viajar en línea recta o bifurcarse de una sección del curso a otra, controlándose el progreso mediante una palanca de mando o un lápiz óptico; el teclado facilita una interacción detallada con el sistema. Gracias a estos dispositivos, el estudiante puede controlar, al nivel deseado, el estudio de los principales sistemas telefónicos.

Además, el disco contiene material gráfico para una futura investigación histórica sobre sistemas de conmutación en telecomunicación. Al final del curso el estudiante recibe la cronología impresa de los eventos.

Un problema que surge con los videodiscos es que no hay ningún sonido durante la exposición de las imágenes estáticas. El PEC está trabajando con dos soluciones: una utiliza un segundo videodisco para emitir el sonido mientras el primero reproduce imágenes fijas, y en la segunda se sintetiza la voz por el ordenador. La primera generación de IVES utiliza una combinación de estas soluciones. No obstante, en el futuro se dispondrá de un videodisco que incorpore sonido e imagen, hoy en desarrollo por los principales fabricantes.

Figura 1
Componentes básicos del sistema de enseñanza interactiva con videodiscos que desarrolla el ITT Programming Education Center.

Desarrollo del curso

El curso de telefonía en IVES se realiza en varias etapas: se resalta sobre todo el diseño interactivo, distribución del disco, contenido visual y preparación de la matriz. Las etapas incluyen:

- Recogida y análisis de información procedente de los fabricantes de sistemas telefónicos.
- Selección de fotografías, diapositivas y secuencias de movimiento para utilizarlas en el curso y para una base de datos de imágenes.
- Desarrollo de un diagrama orgánico del programa que describa las relaciones entre los diferentes medios utilizados (vídeo, audio, ordenador, gráficos, síntesis de voz y texto), el contenido en lecciones y los objetivos del curso. Este diagrama es la representación gráfica de la interacción entre el estudiante, el ordenador y el videodisco, así como de las ramificaciones dentro del programa, el uso de imágenes fijas y en movimiento, sonido y gráficos de ordenador.
- Creación de un guión para el vídeo, el audio y los gráficos de ordenador que integren el programa. En él se detallarán las situaciones, ángulos, iluminación y sucesión de acontecimientos. Se incluye una asignación cuadro por cuadro de todo el material vídeo y audio, según el código estándar de la Society of Motion Picture and Television Engineers y numerando los cuadros del videodisco.
- Diseño y estructura de la base de datos de las imágenes para permitir el acceso y la obtención de información.
- Edición de una cinta maestra de vídeo, de una pulgada, que contenga todo el material, incluyendo cuadros de vídeo y códigos de control.
- Producción de un videodisco de prueba para evaluación final del contenido, los códigos de control, la capacidad de control del ordenador, el programa y la integración de todo el material. Esto permite comprobar la calidad del sistema antes de crear la matriz del videodisco.

Futuro del entorno de aprendizaje interactivo

El futuro inmediato del IVES radica en la integración entre el videodisco, el microordenador y la persona que aprende. La tecnología de videodisco con láser está comenzando a mostrar su potencial como dispositivo de almacenamiento de material educativo. Durante el uso del sistema en el programa de evaluación, se conseguirá conocer más las posibilidades de esta tecnología para su futuro empleo en ITT.

Actualmente, la mayor desventaja de los videodiscos es el alto coste de la producción del vídeo, las dificultades que surgen de la necesidad de fabricarlo en un entorno "limpio", y la imposibilidad de borrar o reescribir información sobre el disco. El primer paso para reducir los costes de fabricación ha sido la posibilidad de leer un disco inmediatamente después de escribirlo, lo que elimina los requisitos de "limpieza". Se han probado también prototipos de discos ópticos borrables.

Armario y equipo para el sistema interactivo de videodiscos.

El impacto real en el proceso educativo ocurrirá cuando la información del videodisco pueda ser borrada y reescrita. Este avance tecnológico reducirá los costes de producción y fabricación, y, junto con el uso de microprocesadores de 16/32 bitios y una programación más avanzada, permitirá un gran desarrollo de los sistemas de enseñanza en esta década. El diálogo entre el sistema y el estudiante estará al nivel de éste y le proporcionará la información requerida, permitiéndole así explorar todos los aspectos de un tema. El resultado será un sistema verdaderamente interactivo, cuyos precursores son la estación de aprendizaje IVES y el curso de telefonía.

Conclusiones

El sistema IVES es un importante avance, no sólo en el mundo de la enseñanza sino también en un amplio campo de aplicaciones, que implican almacenamiento de información posteriormente exhibida a una persona o un grupo, de forma interactiva. Puede prestar un gran soporte al marketing con presentaciones en los puntos de venta y como dispositivo de almacenamiento de rápido acceso aleatorio al material deseado. Puede llegar a integrarse en el puesto de trabajo de los programadores del futuro, con la posibilidad de almacenar y extraer miles de imágenes y miles de millones de octetos de datos. El mismo

sistema se puede utilizar con la misma eficacia en clase, en el despacho de un ejecutivo, en una presentación de marketing, o como archivo. Se puede auto-reconfigurar en función de las necesidades de la persona o de los requisitos de un trabajo, e interactuar con el usuario a niveles de conocimientos preprogramados.

El futuro del videodisco puede abarcar desde la ayuda al entrenamiento en cierto trabajo hasta la formación de una base de datos de imágenes. Específicamente, dentro de ITT se aplicará al entrenamiento en apoyo de los objetivos de ITT sobre formación de programadores.

Bibliografía

1 G. P. Kearsley: The Cost of CAI: A Matter of Assumptions: *Alberta University Division of Educational Research Services Report DERS-01-113,* marzo 1977, Documento ERIC n° IR005777.
2 Premastering/Post Production Procedures for Scotch Videodiscs: Optical Recording Project, St. Paul, 3M, 1981.
3 *Proceedings of the National Conference on Professional Development and Educational Technology,* Washington, DC, 16–18 enero 1980: Association for Educational Communications & Technology, 1980.
4 E. Sigel, M. Schubin y P. Merrill: Video Discs: The Technology, the Applications and the Future: White Plains, Knowledge Industry, 1980, III, 183 págs.

H. Frank Moody nació en Paducah (Kentucky), en 1937. Se graduó BS en ingeniería y educación ocupacional en la Southern Illinois University. De 1958 a 1977 trabajó en plantas de energía nuclear para submarinos y en plantas de turbinas de gas, controladas por ordenador, para buques de superficie en la Marina de EE.UU. Después entró en ITT Telecommunications para dirigir la oficina central y el departamento de entrenamiento de PABX. Tres años después se incorporó al ITT Programming Education Center para trabajar en el desarrollo de tecnologías educativas.

Fenton Wilson nació en New Haven (Connecticut), en 1946. Se graduó BA en inglés por la Universidad de Connecticut en 1973, y MS en educación en la Universidad de Bridgeport en 1977. Durante 1978 estudió ciencias políticas en la Universidad de Jagellonian, Cracovia (Polonia) con una beca Fulbright. De 1979 a 1981 fue el administrador de formación basada en ordenadores en el grupo de Seguros Hartford de ITT. Se unió al Programming Education Center en 1982 como especialista en formación.

Soporte de comunicación ITT/NET para el desarrollo de programas

ITT/NET es un nuevo sistema de comunicación que apoya el desarrollo eficiente y estandarizado de programas en proyectos multinacionales.

S. R. Kimbleton
P. S.-C. Wang
ITT Programming Technology and Development Center, Stratford, Connecticut, Estados Unidos de América

Introducción

Siendo ITT una gran compañía multinacional de telecomunicación, necesita coordinar eficazmente la amplia variedad de actividades de sus unidades operativas en todo el mundo. Ejemplo notable de esta coordinación es la central digital ITT 1240, resultado del esfuerzo combinado de ocho compañías ITT en distintos países.

El caracter multinacional de ITT determina que los expertos en áreas especializadas, como la programación, estén repartidos entre las diversas unidades. Una forma de optimizar la eficacia del esfuerzo en programación es crear un entorno estandarizado, apoyado en los equipos y métodos de formación adecuados.

Una de las ventajas de la estandarización es poder trabajar con un conjunto uniforme de equipos y programas. Evidentemente, esto es difícil de conseguir en una organización tan distribuida como ITT. Sin embargo ITT/NET, nueva red de comunicación de procesadores cuyo prototipo ha diseñado ITT Programming, ofrece muchos de los beneficios esenciales de tal estandarización. Además, el prototipo ITT/NET puede servir de modelo para la red soporte interna de una compañía, proporcionando comunicación de alta calidad de persona a persona por medio de los servicios de "correo de red", acceso remoto a programas y datos, y comunicación entre programas que se ejecutan en sistemas ubicados en centros muy distantes.

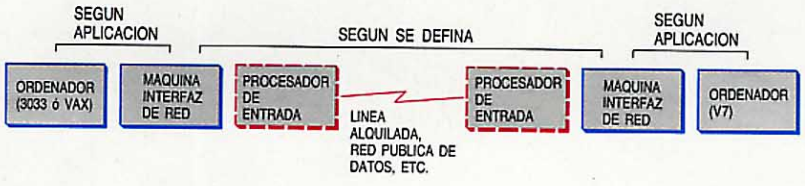


Figura 1
Configuración de red ITT/NET.

Consideraciones de diseño

Los servicios de ITT/NET representan una mejora sustancial respecto a la función de "transporte de bitios", normalmente asociada con una "red de datos". Sin embargo, su realización es difícil ya que requiere interfaces entre ordenadores y usuarios o sus programas. Desde un punto de vista práctico, la conexión de ordenadores a una red exige funciones adicionales para conseguir capacidad de transporte de datos.

Debido a la falta de normalización, los sistemas operativos, sistemas de gestión de base de datos, métodos de acceso de telecomunicación, lenguajes de programación y compiladores, difieren de una unidad a otra en cuanto a nivel y edición. Los distintos entornos para los programas soporte de red (también llamados servicios de red o protocolos de red) ocasionan serios problemas en su desarrollo, prueba e instalación. En lugar de construir un solo programa soporte estándar que sirva para toda la red, se necesita diseñar a propósito y probar cada programa soporte de red siempre que se añada a ésta un nuevo sistema, función o facilidad.

La gestión de configuración (que mantiene control de las diferentes ediciones y versiones de los paquetes de programas) constituye, como es sabido, uno de los mayores problemas de la tecnología de programación. La relativa falta de herramientas dificulta el empleo de la gestión de configuración en la mayoría de las instalaciones. Sin embargo, ello es esencial para un correcto funcionamiento de los programas soporte de red. El problema, pues, está en aplicar tal gestión a estos últimos programas a pesar de no hacerlo en el entorno operacional, que puede de hecho diferir de una compañía a otra.

El diseño de ITT/NET responde a estas consideraciones y a los requisitos básicos siguientes:

- Flexibilidad: la red ha de ser adaptable a la evolución de los requisitos de los usuarios mediante la adición de funciones o facilidades.
- Estrategia multi-proveedor: las distintas circunstancias de cada país imposibilitan obtener el equipo de la red de un único suministrador.
- Independencia: en una gran compañía como ITT, los sistemas de ordenador se instalan o modifican prácticamente a diario; las distintas formas en que los programas se adaptan a las peculiaridades de cada sistema, no deben afectar a la red.
- Aislamiento: las redes nacional (de una compañía) e internacional, deben estar aisladas entre sí; las modificaciones en una no deben afectar a la otra.

Estos requisitos pueden cumplirse si las funciones de soporte de red se pasan de los ordenadores principales a ordenadores separados, denominados máquinas interfaz de red[1,2,3]. Funcionalmente, esto equivale a reemplazar el procesador de entrada tradicional por un procesador de entrada inteligente (Fig. 1).

Requisitos funcionales

Una red para toda la organización debe ofrecer tres clases de funciones: soporte de explotación, soporte de usuario y soporte de bloques constructivos.

Soporte de explotación
Es necesario para optimizar la prevención de errores y de cortes del servicio, detectar estas situaciones anómalas y repararlas rápidamente. La red debe aceptar cargas de trabajo y requisitos variables. Los mecanismos de tarificación deben asegurar que el importe de los servicios sea proporcional a su uso.

Soporte de usuario
En una red de aplicación general, como la ITT/NET, el soporte de usuario consiste normalmente en cuatro servicios importantes: acceso terminal, transferencia de ficheros, correo de red y ejecución distribuida de trabajos.

El acceso terminal permite a un usuario, cuyo terminal está físicamente conectado a un sistema, conectar lógicamente con otro (frecuentemente remoto), accediendo así a bases de datos y servicios de proceso distantes.

ITT/NET optimiza el rendimiento de todo el personal de programación, ofreciéndole un entorno normalizado con soporte de adecuados métodos y materiales educativos.

La transferencia de ficheros permite tanto a usuarios como a programas transferir datos entre ordenadores. Con ello la salida de un trabajo remoto puede transferirse al ordenador local del usuario.

El correo de red permite la comunicación entre usuarios a través de la red.

La ejecución distribuida de trabajos permite al usuario ejecutar un trabajo en un centro remoto, sin tener que anotarlo en ese sitio. Si un centro posee un recurso único, por ejemplo un compilador de aplicación especial, este servicio facilita grandemente la utilización a distancia de tal recurso.

Estos cuatro servicios se han incorporado de diversas maneras en varias redes. La experiencia indica, no obstante, que el correo de red es el más útil. Dos usuarios alejados geográficamente experimentarán cómo su distancia lógica será del orden de la existente entre usuarios en distintos edificios de una misma ciudad.

Soporte de bloques constructivos: arquitectura ITT/NET
Los cuatro servicios básicos indicados son suficientemente fundamentales en la práctica para merecer la denominación de bloques constructivos: útiles al usuario por sí solos, y base para el desarrollo de otros servicios más elaborados. Por ejemplo, estos cuatro bloques simplifican sustancialmente la construcción de un sistema multicentro de gestión de problemas.

Cuando se intenta dar estos tipos de servicios de red en un entorno con múltiples suministradores de equipo, pronto se hace evidente que la red ya no puede verse como un simple medio de transporte de datos al que uno puede conectarse en sus extremos para una aplicación dada.

Se requieren, por el contrario, servicios que cubran los niveles *4* a *7* del modelo de referencia para interconexión de sistemas abiertos (OSI) propuesto por la International Standards Organization[4]; estos niveles corresponden a los que suelen denominarse protocolos de alto nivel. Para dar estos servicios hay que crear un mecanismo estándar de transporte de datos que controle la comunicación entre ordenadores. En la ITT/NET, este mecanismo se llama protocolo universal de transporte, y consiste en un conjunto de servicios de transporte o de extremo a extremo, invocables por programa, que posibilitan la transferencia transparente de datos entre programas de aplicación ejecutables en las máquinas interfaz de red, tales como el sistema de correo, y entre dichas máquinas y las unidades de proceso.

La experiencia en redes existentes inidica claramente que para acertar en la realización de protocolos de alto nivel se necesita una arquitectura unificada. Esta arquitectura debería también expresarse en forma precisa y sin ambigüedades para permitir diversas realizaciones, asegurando al mismo tiempo que el conjunto de servicios representa correctamente la solución completa del sistema.

El principio arquitectónico de ITT/NET es el de niveles, funcionales y estructurales. Los niveles indicados se basan en el modelo de interconexión de sistemas abiertos y en la estructura de los programas de aplicación de comunicaciones del sistema operativo. Ambos aspectos de la arquitectura de ITT/NET están íntimamente relacionados; cada nivel funcional corresponde a un componente estructural, y por tanto ofrece una base natural para modularizar los programas que realizan las funciones.

La jerarquía de protocolos de ITT/NET comprende cuatro niveles funcionales. Cada nivel, excepto el más bajo, utiliza las funciones del inmediato inferior y añade una o más funciones para ser usadas por el nivel superior siguiente. El nivel más bajo consiste en paquetes de comunicación, como las facilidades de red de IBM, que proporcionan los medios básicos de transmisión de datos (la red de comunicación). El nivel siguiente especifica medios para la transferencia de datos desde la máquina interfaz de red origen a la máquina interfaz de red o unidad de proceso de destino. Este protocolo universal de transporte de ITT/NET se usa en todos los servicios de nivel más alto como vehículo común de comunicación de ordenador a ordenador.

El tercer nivel contiene los protocolos que prestan servicios soporte generales, como acceso a ficheros, ejecución distribuida de trabajos, acceso terminal, y correo de red. Estos protocolos sirven también de base para construir los protocolos superiores de desarrollo multicentro. Un ejemplo de éstos sería un sistema distribuido de gestión de problemas que utilizara el servicio de correo de red para informar sobre dichos problemas, los servicios de acceso terminal y transferencia de ficheros para reproducción e identificación de los mismos, y la ejecución distribuida de trabajos (junto con otras facilidades como un sistema local de gestión de base de datos) para el rastreo y control de tales problemas (Fig. 2).

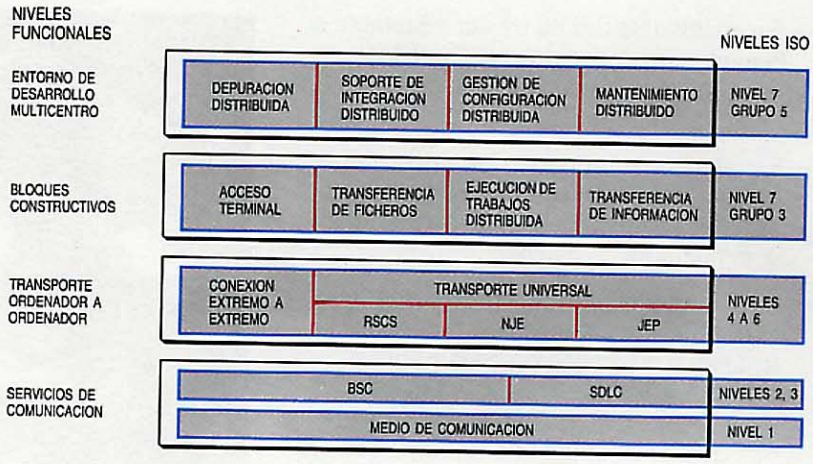


Figura 2
Jerarquía funcional ITT/NET.
BSC - control binario síncrono
JEP - programa de entrada de tarea
NJE - entrada de tarea en la red
RSCS - superficie de comunicación de subsistemas remotos
SDLC - control síncrono de enlace de datos.

Las ventajas de la estructuración en niveles son bien conocidas: mejora la gestión e independencia de dichos niveles mediante la agrupación funcional y la separación de funciones en capas diferentes. Esto es especialmente importante en ITT/NET por varias razones. En primer lugar, la flexibilidad de reemplazar funciones de un nivel dado por otras equivalentes, sin afectar con ello a los otros niveles. En segundo lugar, el marco que se establece para desarrollar nuevos protocolos. Así, las dos áreas críticas de crecimiento de ITT/NET— mejoras en los protocolos y construcción de aplicaciones distribuidas — pueden planificarse y realizarse en forma ordenada. Finalmente, y tal vez lo más importante, dada una base de programación de la máquina interfaz de red, la estructuración en niveles no sólo impone la partición funcional entre protocolos sino que establece el marco para la modularidad del diseño.

Estructuralmente, cada protocolo de nivel de usuario de ITT/NET se organiza en tres componentes lógicos, caracterizados por el tipo de operaciones que realizan:

– *Componente soporte de usuario* (USC): interpreta las órdenes de usuario, accede a los ficheros de éste y en general le facilita funciones de soporte.

– *Componente soporte de protocolo* (PSC): realiza funciones no visibles directamente por el usuario (p. ej., almacenamiento intermedio y tratamiento de atributos de fichero).

– *Componente soporte de transporte* (TSC): transfiere información invocando y actuando de interfaz con los métodos de acceso al ordenador principal, como el VTAM (Virtual Telecommunication Access Method) y el RSCS (Remote Spooling Communication Subsystem).

La figura 3 muestra estos tres componentes y el flujo de control y datos entre ellos, así como la correspondencia entre los niveles estructurales antes debatidos y los distintos componentes funcionales. La caja de servicios de comunicación en este diagrama realiza los servicios ITT/NET de nivel *1* y cubre la comunicación TSC-TSC. Sin embargo, ya que los paquetes de proveedores que realizan el nivel *1* son diversos y normalmente incompatibles, el nivel *2* (protocolo universal de transporte) proporciona el servicio de transporte unificado y válido a escala de red. Estructuralmente este nivel es el interfaz TSC-PSC; también incluye los pre y post-procesadores asociados con este interfaz (p. ej.: empaquetamiento de datos en forma aceptable por el TSC). Este servicio de transporte a su vez constituye la base para la realización de otros protocolos ITT/NET que ofrecen el interfaz USC-PSC y realizan otros procesos específicos de soporte directo al usuario.

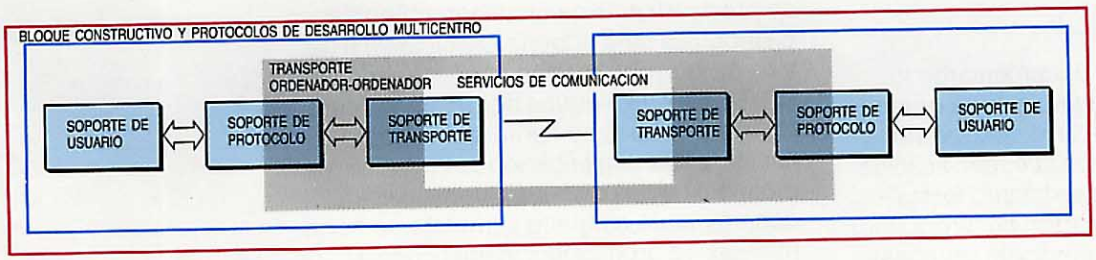


Figura 3
Jerarquía estructural ITT/NET.

Proceso de programación ITT/NET

Pequeñas diferencias en los programas distribuidos internacionalmente pueden conducir a problemas importantes en su mantenimiento y mejora. La eliminación de errores en los programas es también notablemente más costosa que en el caso de un centro único. Por estas razones, el método de programación usado en el desarrollo de ITT/NET emplea técnicas formales en la especificación de protocolos, y procedimientos rigurosos para probar y verificar las realizaciones. Una aplicación típica del proceso de programación fue la producción del protocolo de correo de red, que permite a un usuario crear, editar, archivar y distribuir informes a otros usuarios. El proceso constó de cinco etapas: definición, diseño detallado, producción, disponibilidad limitada y disponibilidad general. No obstante, por tratarse de un prototipo, sólo se consideran aquí las tres primeras.

Definición. Esta fase consistió en: análisis de requisitos, especificación funcional, diseño de alto nivel, y planificación de pruebas de alto nivel. Con objeto de conseguir rigor y precisión, se llevó a cabo un análisis exhaustivo de las facilidades de correo de red existentes y de las necesidades del usuario. De ello se obtuvo un núcleo de órdenes como base para la especificación funcional del servicio en cuestión. El diseño de alto nivel se expresó mediante un lenguaje formal de descripción de procesos[5], como colección de programas, ficheros y módulos para realizar estas funciones. Finalmente, se estableció un plan de pruebas funcionales para detectar desviaciones respecto al comportamiento especificado. El resultado de esta fase fue una especificación del sistema de correo de red en el lenguaje de descripción de procesos, y un plan de pruebas funcionales que sirviera de base para la etapa siguiente.

Diseño detallado. El diseño inicial se amplió y reorganizó hasta alcanzar un nivel de detalle apropiado para la especificación de los módulos. La especificación en lenguaje de descripción de procesos sirvió también como base para obtener el plan detallado de pruebas que verificó la corrección lógica de los módulos: se seleccionó un conjunto de datos de prueba y se formuló la representación de los módulos por tablas de decisión, para confirmar que todas las ramas e instrucciones se probaban al menos una vez.

Producción. Se codificó el diseño detallado, se pasaron pruebas funcionales y de mó-

Tabla 1 – Situación actual del correo de red ITT/NET

Función

Ofrecer un conjunto integrado de facilidades para la preparación, distribución y almacenamiento de textos (actualmente soporta la transferencia BSC de informes VM a VM y VM a VAX/VMS).

Características principales

2,1 k-líneas de PL/I (13 módulos),
1,9 k-líneas de ensamblador IBM370 (17 módulos).

Sin errores conocidos, se precisan más pruebas intensivas.

Instalado en tres 4300 en ITT Shelton y Stratford.

Nivel de utilización: todos los usuarios de ordenadores de ITT Programming.

Facilidades

Preparación, almacenamiento y entrega de textos en un paquete.

Integrado en el entorno actual de proceso de datos.

Puede manejar diversas fuentes de datos.

Puede interconectarse con sistemas existentes de documentos/bibliotecas.

Extensiones futuras

Otros interfaces (VM a MVS y VM a VAX/UNIX).

Cifrado y descifrado de datos.

Archivo y recuperación de informes.

Contestación y reenvío automáticos.

Servicios centralizados de almacenamiento y archivo.

dulo, y se analizaron y documentaron los resultados. Se realizó también una prueba de la corrección lógica del diseño, verificando que éste satisfacía la especificación funcional del correo de red, tanto formalmente (mediante verificación por programa) como empíricamente (mediante pruebas).

Soporte de campo para ITT/NET

Como antes se indicó, la descarga de la mayor parte de los programas de los servicios soporte de red desde el ordenador principal a la máquina interfaz de red, da una base mucho mejor para la gestión de configuración de los programas que realizan los servicios soporte. Concretamente, los interfaces con la referida máquina que se ejecutan en el ordenador principal son mucho más sencillos que los programas necesarios para llevar a cabo el servicio completo de la red desde dicho ordenador. Con ello, se reduce significativamente la modificación del entorno operativo existente a ejecutar en la unidad principal.
La mayor parte de las tareas de soporte y mejora de los servicios de red puede centralizarse, aumentando así la productividad. Además, para participar en la red, los distintos centros no precisan de expertos locales en diseño y optimización de redes. Finalmente, el soporte central de los servicios de red se adapta bien al soporte centralizado de las funciones de transporte ofrecido por la arquitectura de la red. Ello hace que los usuarios acudan a una organización única para la prevención, detección y solución de los problemas.

Aunque dicha descarga ofrezca los beneficios indicados, su realización requiere un desarrollo adicional importante para soportar la carga centralizada de programas, así como la supervisión y gestión de los servicios y del acceso a la red. Actualmente ITT Programming está desarrollando una arquitectura básica para el soporte de este proceso.

Conclusiones

Los prototipos actualmente existentes en ITT/NET, transferencia de ficheros y correo de red (tabla 1), muestran la factibilidad de un soporte eficaz de comunicación para los proyectos multinacionales, como el desarrollo de la central digital ITT 1240. Sin embargo, se precisan otras muchas herramientas distribuidas para soporte del ciclo total de vida de la programación. Los interfaces de ITT/NET y sus bloques constructivos funcionales básicos se han diseñado para facilitar estas amplificaciones futuras. En los próximos años, el desafío consistirá en desarrollar un conjunto completo de herramientas de programación multicentro para un entorno de desarrollo integrado, al servicio de los proyectos de programación de ITT repartidos por todo el mundo.

Referencias

1 L. G. Roberts y B. D. Wessler: Computer Network Development to Achieve Resource Sharing: *Proceedings of the Spring Joint Computer Conference,* Atlantic City, mayo 5–7, 1970, American Federation of Information Processing Societies Conference Proceedings, volumen 36, 1970, págs. 543–549.

2 E. Feinler y J. Postel: ARPANET Protocol Handbook: Menlo Park, Stanford Research Institute, Network Information Center, 1976, IV, 362 págs.

3 S. R. Kimbleton, P. Wang y B. W. Lampson: Applications and Protocols: Distributed Systems Architecture and Implementation: *Lecture Notes in Computer Science,* Nueva York, Springer-Verlag, 1981, volumen 105.

4 International Organization for Standardization, ISO/TC 97/SC 16: Reference Model for Open Systems Interconnection, noviembre 1980.

5 R. C. Linger, H. D. Mills y B. I. Witt: Structured Programming, Theory and Practice: Reading, Massachusetts, Addison-Wesley, 1979, XIV, 402 págs.

Steve Kimbleton obtuvo su PhD en la Ohio State University. Comenzó a trabajar en programación en IBM, en el área de control de procesos para la industria papelera. Posteriormente trabajó para el USC/Information Sciences Institute, la Rand Corporation y la Universidad de Michigan, donde investigó en ordenadores y su intercomunicación. El Dr. Kimbleton fue jefe de protocolos de red para el National Bureau of Standards antes de entrar en ITT Programming en Stratford, donde actualmente es director de arquitecturas lógicas de red.

Pearl S.-C. Wang nació en China en 1941. Recibió los grados BSc (química) y PhD (química teórica) en la Universidad Nacional de Taiwan y en la Universidad de Wisconsin en 1963 y 1968, respectivamente. Después de trabajar para la Computer Sciences Corporation, el National Institute of Health y el National Bureau of Standards, entró en ITT Programming en 1980. La Dra. Wang dirige actualmente el desarrollo de protocolos de red en ITT Programming.

Entorno de programación CHILL para el ITT 1240

CHILL, el lenguaje de alto nivel diseñado por el CCITT para aplicaciones de conmutación telefónica, forma la base del entorno de programación para el ITT 1240.

T. Haque
ITT Europe Programming Support Centre, Harlow, Inglaterra

Introducción

Antes del año 1980, y de acuerdo con un estudio realizado por el CCITT, había en todo el mundo 1500 programadores CHILL, de los cuales 800 trabajaban en ITT. Este simple hecho subraya el audaz compromiso de ITT y la confianza que puso en apoyar el lenguaje CHILL. Las ventajas y desventajas de usar CHILL fueron sopesadas cuidadosamente; se reconoció, desde muy pronto, que se podía obtener una ganancia muy significativa, a pesar de que entonces el lenguaje no estuviera aún estabilizado. Este interés dio como resultado unas herramientas capaces de soportar el uso del CHILL, satisfaciendo así las diversas necesidades de la Compañía. Este artículo trata principalmente de la aplicación del CHILL al desarrollo de la central digital ITT 1240.

Programación en las telecomunicaciones

A finales de los años 60 el CCITT emprendió unas primeras investigaciones sobre la programación en sistemas de control por programa almacenado (SPC). Se percibía ya que el creciente uso de sistemas de conmutación SPC significaba que la programación jugaría un papel dominante en el desarrollo de futuras centrales telefónicas.

Durante el periodo de estudio 1973 a 1976, el CCITT empezó a trabajar en la creación de una familia de lenguajes para sistemas SPC:

- el CHILL, lenguaje de alto nivel para conmutación telefónica
- el SDL, un lenguaje de especificación y descripción
- el MML, lenguaje hombre-máquina del CCITT.

La decisión del CCITT de seguir adelante con la definición del lenguaje CHILL se fundamentó en dos razones principales. Primero, la necesidad de proveer un solo lenguaje estándar que cubriese todos los productores y usuarios de sistemas SPC. Segundo, consolidar la fortaleza de los lenguajes existentes en un solo lenguaje, con especial atención a las necesidades de la programación para sistemas SPC. ITT ya empleaba diversos lenguajes de alto nivel para sistemas SPC, y se comprendió que la normalización sería muy beneficiosa.

El CCITT evaluó en principio 27 lenguajes existentes. Sin embargo, en 1974 se concluyó que ninguno de esos lenguajes podría satisfacer los criterios establecidos. Consecuentemente, se formó un grupo llamado "equipo de especialistas" para definir un nuevo lenguaje.

En 1977 el "equipo de especialistas" fue reemplazado por un "foro de realizadores", al que se le asignaron las tareas de llevar a cabo las propuestas preliminares, de recoger experiencia y de informar sobre los resultados. Hacia finales del año 1979 se tenía ya completada la primera propuesta de lenguaje; en 1980, después de algunas mejoras editoriales, se presentó en la Asamblea Plenaria del CCITT, ésta aprobó la definición del lenguaje (noviembre de 1980), y fue entonces publicada como Recomendación Z.200.

Los requisitos que tuvo que cumplir el CHILL se referían a las aplicaciones asociadas con la telefonía: tratamiento de llamadas, prueba y mantenimiento, soporte en-línea y fuera-de-línea, etc. Sin embargo, dados estos requisitos y el reconocimiento de que los sistemas de conmutación telefónica eran cada vez mayores y más complejos, los requisitos del lenguaje habían de tener otra dimensión. El CHILL, de esta forma, tenía que ser un lenguaje para "programación en profundidad", necesariamente caracterizado por su mayor fiabilidad, capacidad de generar código objeto con gran eficiencia, facilidad de aprendizaje, flexibilidad y potencia, al tiempo que preservara y favoreciera una programación modular y estructurada.

El CHILL está construido para lograr fiabilidad activa y pasiva. La fiabilidad pasiva se consigue al diseñar las construcciones del lenguaje de forma que ante todo se evite la introducción de errores. Las construcciones del lenguaje son flexibles y eficaces,

facilitando el modelado de sistemas en una forma natural y potenciando la programación modular y estructurada. En este contexto destacan conceptos tan importantes como procesos, módulos y procedimientos.

La fiabilidad activa se logra al ofrecer métodos para una comprobación total de consistencia, siendo de destacar el control sobre la visibilidad y el control sobre el modo. Las reglas de visibilidad definen qué nombres son accesibles en las diversas partes del programa. Las reglas relativas a los modos permiten asegurar que los nombres se utilizan de acuerdo con sus propiedades, también definidas. Para lograr la necesaria fiabilidad sin reducir la eficiencia, la mayoría de las revisiones pueden hacerse estáticamente en el momento de la compilación.

Hay unas pocas reglas, las denominadas "condiciones dinámico-semánticas del CHILL", que no pueden comprobarse estáticamente. La violación de una de estas reglas da lugar a una excepción en la ejecución del programa. Las excepciones pueden también ser originadas explícitamente por el usuario. Cuando ocurre una excepción, el usuario puede recuperar el control del programa si dispone del correspondiente operador al cual transferir el control.

El CHILL, como lenguaje, es independiente de la máquina. Sin embargo, una aplicación particular puede contener características de lenguaje orientadas a la aplicación (por ejemplo, para comunicar con programas y equipos ya existentes).

Programación en profundidad

La central digital ITT 1240 es el mayor proyecto de desarrollo acometido hasta ahora por ITT, y se caracteriza por lo que suele denominarse programación "en profundidad". Además, el desarrollo del producto es verdaderamente multinacional, involucrando a personal con múltiples especialidades en varios países. El ciclo de vida del producto suele ser del orden de décadas y comprende muchas fases, durante el desarrollo y después de la instalación. Como cada central telefónica satisface requisitos diferentes, no hay dos instalaciones idénticas.

El desarrollo del ITT 1240 no lo realiza una sola, sino varias compañías de ITT, cada una con sus características propias en cuanto a gestión, experiencia tecnológica e intereses comerciales. Los requisitos de prueba para un sistema SPC grande y moderno son rígidos y consumen mucho tiempo; la distribución geográfica de las compañías de ITT participantes en el desarrollo impone métodos de prueba continuos e incrementales, a todos los niveles (desde una pequeña unidad de programación a una central entera), que se integren rigurosamente en la estrategia de diseño y realización. Además, como resultado de la evolución tecnológica, mejoras de diseño y correcciones, la necesidad de un tratamiento de cambios rápido y sincronizado obligó a establecer una red de ordenadores entre los distintos centros de diseño. Todavía más importante fue disponer de un servicio activo y automatizado de control de cambios para el seguimiento y actuación sobre los cambios pendientes.

Por consiguiente, la coordinación de estos grupos nacionales para alcanzar el objetivo común exigió un entorno de programación CHILL fácilmente asimilable y uniforme y el reconocimiento de que había que prestar grados de atención diferentes al entorno de desarrollo de programas y al de producción de programas. Mientras que el entorno de desarrollo de programas tenía que proteger la capacidad de los programadores contra la acumulación de trabajos extra que pudieran solicitárseles, el entorno de producción de programas tenía que ofrecer rigor y disciplina para asegurar la extremada fiabilidad en la construcción y mantener un eficaz control de la configuración.

Bloques constructivos de la programación ITT 1240

La arquitectura de la central ITT 1240 (Fig. 1) puede servir para explicar el entorno de programación CHILL, el cual enlaza íntimamente con la arquitectura de diseño de programas del ITT 1240.

La central comprende esencialmente diversos módulos de red que proveen las funciones telefónicas. Estos módulos de red están distribuidos por toda la central, siguiendo los criterios de configuración y dimensionamiento de la misma.

Cada módulo es autónomo en sí mismo y se le puede ver como un terminal de red completo. Consta de los siguientes componentes básicos: un microprocesador, memoria, un interfaz terminal y, usualmente, el equipo de aplicación asociado. El interfaz provee las funciones de comunicación a nivel físico necesarias para la red digital de conmutación. El tamaño de la memoria varía entre 256 k-octetos y 1 M-octeto, según la aplicación. El equipo asociado depende también de la función a realizar. El microprocesador proporciona la potencia de cálculo necesaria para las funciones relativas a la aplicación y para las actividades de control de la red. El micropro-

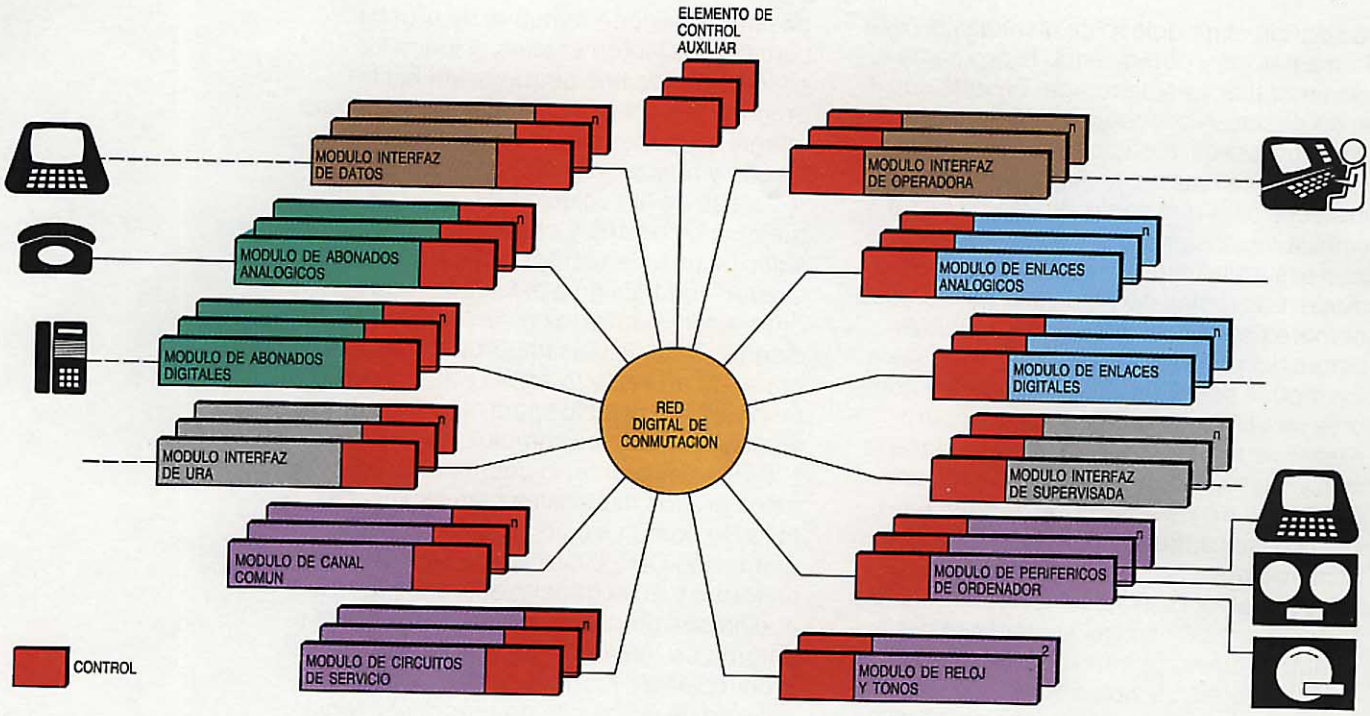


Figura 1
Arquitectura de la central digital ITT 1240.
URA - unidad remota de abonado.

cesador e interfaz terminal son iguales para todos los módulos de la red.

Al conjunto de microprocesador, su memoria y el interfaz terminal se le denomina elemento de control del módulo terminal. Es importante insistir en que la distribución de los elementos de control, junto con la especial estructura de la red digital de conmutación, caracteriza la arquitectura de la central digital ITT 1240.

En cada elemento de control tienen que cargarse los programas de aplicación. La asignación de estos programas y la forma en que son producidos es tarea del entorno de programación CHILL. De hecho, los programas de aplicación son máquinas soporte del ITT 1240 (máquinas de mensajes finitos y máquinas soporte del sistema) pero pueden considerarse, desde una perspectiva de desarrollo, como componentes de programación comunes y específicos[1]. Estas máquinas soporte, ya sean comunes o específicas, son los bloques constructivos, y en virtud de su diseño pueden ser configuradas en elementos de control. Cuando se asignan y cargan en la memoria de un elemento se denominan segmento genérico de carga (GLS). La base de datos en-línea es el segmento de carga de datos (DLS).

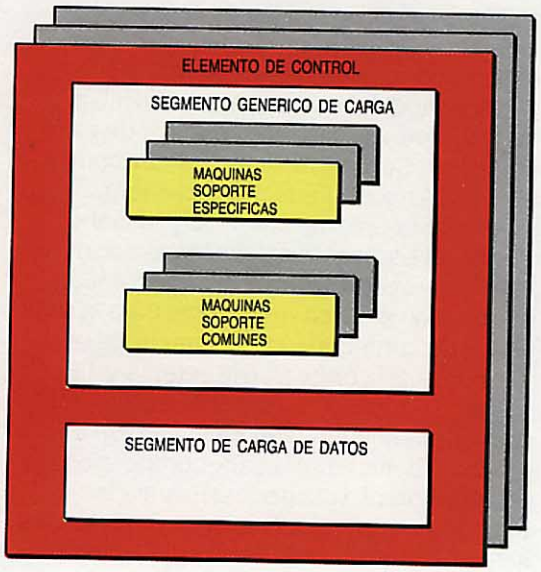


Figura 2
Disposición de la memoria en un elemento de control.

Configurabilidad

El diagrama de disposición de memoria (Fig. 2) muestra un segmento de carga y su segmento de carga de datos asociado; éste, una vez poblado, da soporte tanto a las máquinas comunes como a las específicas. El sistema operativo y el sistema de gestión de la base de datos permiten la comunicación entre las máquinas soporte y la gestión de datos en sí misma.

La configurabilidad en segmentos de carga se explica en la figura 3. Las máquinas soporte, tanto comunes como específicas, se configuran y mantienen en bibliotecas que pertenecen a un cierto mercado. Dependiendo de la configuración particular de la central, se combinan los apropiados segmentos de carga genéricos para construir el paquete específico de la central. El segmento de carga de datos correspondiente se produce por separado, para cada uno de los elementos de control de la central, por el sistema soporte para producción de programas.

Captación del diseño de alto nivel

Se asimila el diseño de alto nivel del sistema ITT 1240 con el fin de asegurar que la construcción de las máquinas soporte sea consistente y se adhiera a la estrategia de diseño del ITT 1240.

Para los fines del presente artículo, esto puede ser considerado en dos partes que son el nexo principal de unión entre la fase de diseño de alto nivel y la fase de diseño detallado/realización en el ITT 1240. Este interfaz se mantiene en dos bases de datos y se utiliza por los procesadores de lenguaje fuente en la fase de realización para asegurar que las máquinas soporte son conformes al diseño. Resumiendo, el contexto, desde el punto de vista del diseño de alto nivel, permite:

- Describir todos los interfaces entre programas del sistema: mensajes (asíncrono), llamadas a función externa (síncrono), componentes e interrelaciones de la base de datos.
- Aislar la información de diseño de alto nivel frente a cambios en el momento de realización.
- Comprobar, durante la producción de programas, que se mantiene la integridad de todos los interfaces.

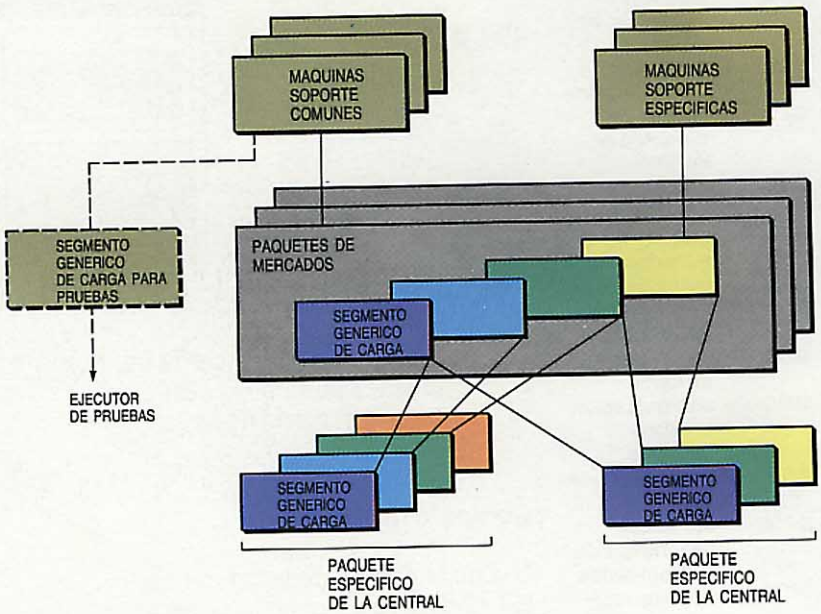
Las dos partes de la base de datos, que reflejan el diseño de alto nivel, son la biblioteca descriptiva de los componentes de la programación y la base de datos distribuida del ITT 1240 (metabase). La biblioteca contiene toda la información de los interfaces definidos por el diseño de alto nivel y se almacena en la metabase de datos, ya que los datos del ITT 1240 son parte de dicho diseño y su gestión está separada del proceso de programación normal. Esta base de datos relacional se utiliza por el programador CHILL en el momento de traducir el programa fuente mediante el pre-procesador Multipol.

Gestión de la base de datos

La naturaleza distribuida del control de la central conduce a asignar los datos entre los elementos de control. Para ocultar esta distribución a los programadores, cada elemento de control contiene un segmento de carga de datos (DLS) y un sistema de gestión de la base de datos. Los datos contenidos en cada DLS son generados, para cada elemento de control, durante la producción de programas. El DLS contiene información sobre las relaciones específicas de las máquinas soporte que han de ser cargadas en ese elemento de control. La separación entre datos y código hace posible imponer un control sistemático de los datos. Las ventajas son significativas al reducir la redundancia en datos y aumentar su integridad; pueden además resolverse los conflictos entre requisitos de los datos. Este tipo de gestión también dispensa al programador de conocer la organización física de los datos y cómo extraerlos, proveyendo así un mecanismo de seguridad sobre los datos.

Dentro del entorno CHILL, un lenguaje especial de manipulación de datos forma el interfaz con el sistema de gestión de la base de datos. No se trata de un lenguaje de procedimientos contenido en sí mismo, sino que está diseñado para existir dentro del lenguaje CHILL. Es decir, las instrucciones del CHILL y del lenguaje de manipulación de datos se entremezclan en los programas de aplicación. Las órdenes dadas en este lenguaje son convertidas a instrucciones CHILL por el preprocesador.

Figura 3
Bloques constructivos del ITT 1240: configurabilidad.



Entorno de programación CHILL del ITT 1240

Tres principales sistemas soporte juntos forman un entorno integrado de producción de programas para el ITT 1240, según la figura 4: el sistema soporte para desarrollo de programas, el de ingeniería de aplicación y el de producción de programas. El primero de ellos alberga al entorno CHILL de desarrollo de programas, y los otros dos se describen más adelante. Se producen máquinas soporte comunes y específicas, ubicándolas en bibliotecas de segmentos

genéricos de carga que se dirigen a áreas de mercado (Fig. 3).

El contexto del ITT 1240 es una colección de descripciones de componentes que proveen o definen un entorno de programación dentro del cual pueden crearse las máquinas soporte y producirse una determinada central. El contexto se utiliza a lo largo de todo el ciclo de vida del desarrollo de programas; no sólo contiene información sobre el diseño de alto nivel sino también

(POL) para asegurar que la realización de las máquinas soporte cumplía con el citado diseño, definido en la biblioteca de componentes de programación y en la base de datos.

En consecuencia, se definieron unos POL que soportaran los diferentes aspectos de la manipulación de datos, de la descripción de interfaces y de las relaciones entre los componentes de programación. En su forma fuente, el programa del

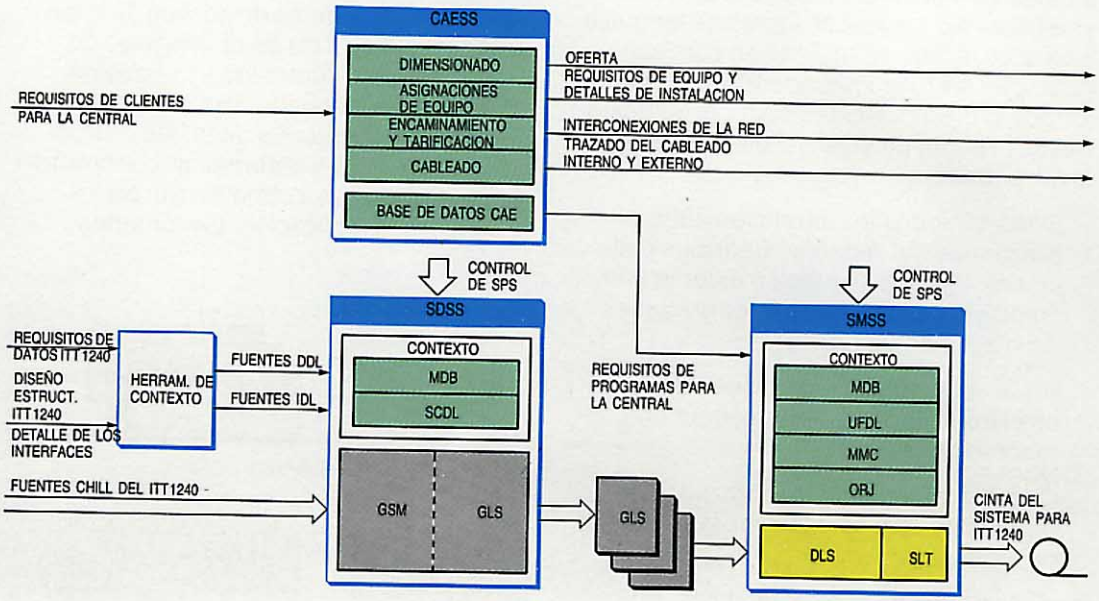


Figura 4
Procesos de producción de programas ITT 1240.

CAE - ingeniería de aplicación
CAESS - sistema soporte de ingeniería de aplicación
DDL - lenguaje de definición de datos
DLS - segmento de carga de datos
GLS - segmento genérico de carga
GSM - máquina soporte del módulo fuente genérico
IDL - lenguaje de descripción del interfaz
MDB - metabase de datos
MMC - comunicación hombre-máquina
ORJ - peticiones por operador
SCDL - biblioteca descriptiva de componentes de programación
SDSS - sistema soporte para desarrollo de programas
SLT - cinta del sistema
SMSS - sistema soporte para producción de programas
SPS - hojas de producción de programas
UFDL - biblioteca descriptiva de ficheros del usuario.

sirve de base para la ingeniería de diseño para clientes, a partir de la cual puede realizarse la ingeniería de aplicaciones.

Contexto del ITT 1240

Los diseñadores de la central digital ITT 1240 proporcionan información al más alto nivel, incluyendo el diseño de los datos. Esta información se introduce en el sistema soporte para desarrollo de programas por medio de un conjunto de herramientas del contexto que traducen los requisitos del diseño de alto nivel a instrucciones del lenguaje de definición de datos o del lenguaje de descripción de interfaces. Como se muestra en la figura 5, estas instrucciones después se traducen y mantienen en la base de datos y en la biblioteca que describe los componentes de la programación, las cuales son elementos principales del contexto del ITT 1240.

Traducción de la fuente CHILL

El diseño de alto nivel necesitó el soporte de lenguajes adaptados a los problemas

ITT 1240 consiste en instrucciones CHILL e instrucciones en POL. La función del preprocesador (llamado Multipol por su capacidad de tratar varios POL) es procesar el programa fuente y, con la ayuda de la biblioteca descriptiva de los componentes y de la base de datos, reemplazar todas las instrucciones POL por instrucciones CHILL (Fig. 6). Tres lenguajes POL trata el preprocesador para producir instrucciones CHILL que luego puedan compilarse:

– *Lenguaje de estructura de programa (PSL),* utilizado para proveer de estructura uniforme a los programas CHILL del ITT 1240. Lo más importante es que da a conocer al preprocesador cuál descriptor (máquina de mensajes finitos o máquina soporte del sistema) emplea el programa. El Multipol usa este descriptor para acceder a la biblioteca de componentes y obtener los detalles de interfaz para este programa. Como resultado, el Multipol es capaz de acceder a la base de datos y generar las instrucciones CHILL para todos los modos y declaraciones que soportan la comunicación (vía mensajes) con otras máquinas de mensajes finitos o máquinas soporte del sistema.

– *Lenguaje de manipulación de datos (DML),* que uniformiza las solicitudes de datos a la base de datos ITT 1240. El modelo que define los derechos de acceso y el submodelo de ese programa son extraídos de la base de datos y utilizados para comprobar que las solicitudes expresadas en instrucciones del lenguaje de manipulación de datos son válidas. Si una solicitud es válida, entonces se generan las apropiadas instrucciones CHILL, incluyendo una llamada al sistema de gestión de la base de datos.

– *Lenguaje del núcleo del sistema operativo (OSNL),* por el cual se accede al sistema operativo para ejecutar las funciones de tratamiento de mensajes. Al encontrar una instrucción del OSNL, se consulta a la biblioteca descriptiva de componentes, y si es válida la función para el mensaje identificado, entonces se generan las apropiadas instrucciones CHILL, incluyendo una llamada al núcleo del sistema operativo.

Compilación CHILL

El compilador acepta programas que contienen uno o más módulos CHILL y produce un código objeto no montado, el cual se monta posteriormente. Las máquinas soporte son unidades de producción y se emplea el mecanismo *grant/seize* para resolver referencias al montar el programa. En paralelo, se dispone de medios para montar juntos un módulo CHILL y otro ensamblador. Los programas CHILL se compilan y montan; el resultado se almacena en el formato reubicable de ITT. Los módulos genéricos de programación o máquinas soporte se mantienen en una biblioteca y pueden montarse juntos, formando segmentos de carga para elementos de control específicos.

Se llama proceso GSM (Fig. 6) a la unión del proceso de traducción de la fuente con el de compilación CHILL en una máquina soporte, y se le automatiza mediante el sistema soporte para desarrollo de programas (SDSS). El proceso lo inicia una biblioteca de SPS (hojas de producción de programas) que proporciona las listas de elementos de entrada y los detalles del proceso.

Proceso de producción de un segmento genérico de carga

El sistema de montaje se usa para ubicar una selección de máquinas soporte en un módulo denominado segmento genérico de carga (GLS), destinado a cargarse en un elemento de control específico. El proceso de producción es también automático, merced al SDSS.

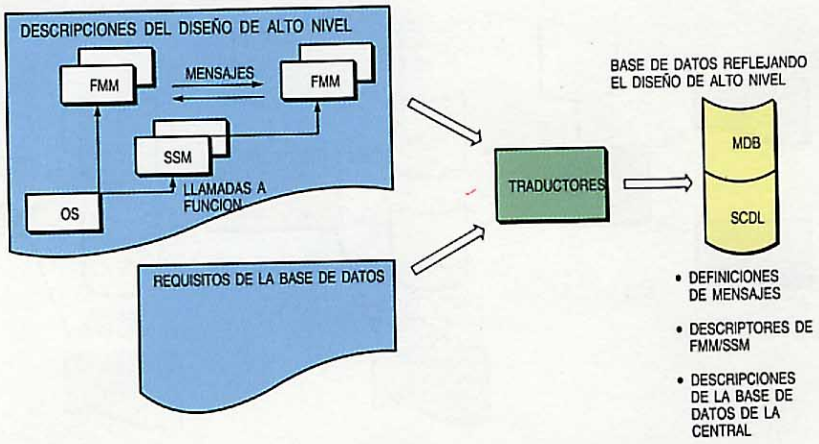


Figura 5
Captación del diseño de alto nivel: preparación del contexto ITT 1240.
FMM - máquina de mensajes finitos
OS - sistema operativo
SSM - máquina soporte de sistema.

Nivel de prueba del CHILL

Se requieren pruebas a nivel de módulo, de la integración y de sistema durante el ciclo de vida de la programación. El TEX (programa ejecutivo para pruebas) es un programa de depuración a nivel simbólico, diseñado como herramienta de prueba para programas CHILL. Puede ser usado por lotes (batch) o en modo interactivo. En ambos casos, los probadores emplean el lenguaje de prueba para controlar y observar las pruebas (Fig. 7). Dicho lenguaje es similar al CHILL y da soporte a funciones tales como la visualización y modificación de datos, la alteración del flujo del control del programa y corrección temporal de códigos erróneos. Para el ITT 1240 es usual probar una máquina soporte como parte de la actividad de pruebas de módulo unitario. Dadas las características de la máquina soporte, puede establecerse fácilmente su entorno exterior. El proceso de producción de segmentos genéricos de carga se utiliza para aislar una máquina soporte y su entorno de prueba, de modo que pueda probarse unitariamente mediante el SDSS.

Sin embargo, para los niveles de pruebas de integración y de sistema se requiere una

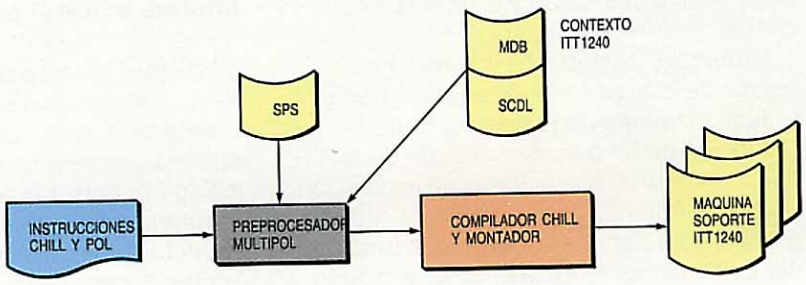


Figura 6
Proceso GSM: máquina soporte del ITT 1240.

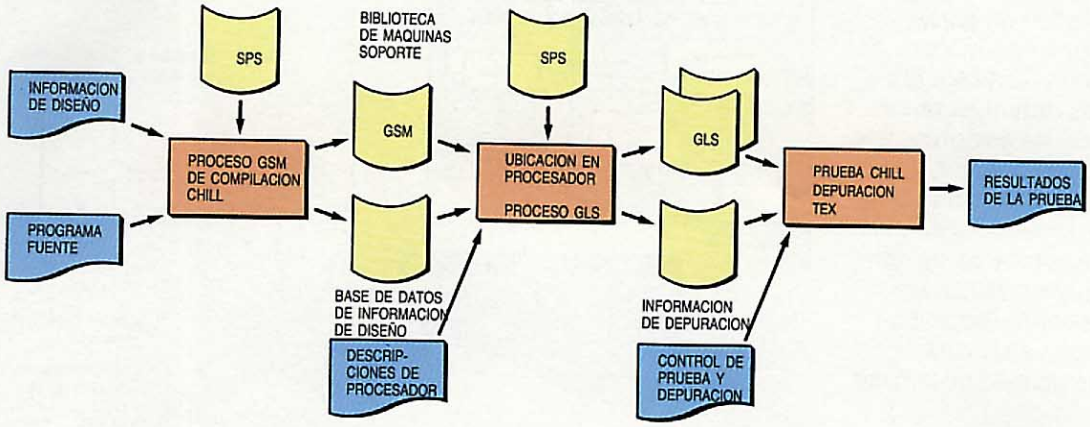


Figura 7
Sistema soporte para desarrollo de programas: prueba de programas CHILL. TEX - ejecutor de pruebas.

capacidad mayor. Los medios de prueba en laboratorio (LTF) dan capacidad para probar la central digital ITT 1240, no sólo a nivel unitario, sino a nivel de varias máquinas soporte a la vez, de un elemento de control, y de un grupo de elementos de control funcionalmente relacionados. Como el sistema ITT 1240 se basa en procesadores distribuidos, los medios de prueba admiten también más de un usuario con el fin de probar varios procesadores de forma concurrente. Además, el LTF provee un mecanismo controlado dinámicamente que puede asignar los recursos de prueba según dicten las necesidades del probador. El TEX se ejecuta también bajo el control del LTF, permitiendo la prueba simbólica y la prueba en bajo nivel (sobre direcciones específicas).

Otros sistemas soporte

Aunque en sentido estricto no pertenezcan al entorno CHILL, citaremos brevemente otros dos sistemas soporte usados en la programación ITT 1240.

Sistema soporte de ingeniería de aplicación (CAESS)
Este sistema soporta las actividades de ingeniería para la manufactura e instalación de una central digital específica, o de ampliación a una ya existente (Fig. 4). En consecuencia, este sistema soporte ha sido diseñado para utilizarse en preparación de ofertas, facilitando la composición del equipo, su distribución y el coste de cada central.

El objetivo principal es minimizar los costes de la ingeniería de aplicaciones, reducir el tiempo de respuesta entre la recepción de un pedido y el corte de la central, y acortar los periodos de instalación y prueba. Además de ayudar a la gestión de ofertas, este sistema soporte lleva a cabo el dimensionamiento de la central, el diseño de la distribución de equipos y la interconexión. También genera una lista completa de documentos por central.

En particular, y desde el punto de vista de la programación CHILL, se generan aquí todos los requisitos necesarios para poblar los DLS (según se definen en la metabase de datos). Estos requisitos son el interfaz con el otro sistema soporte (SMSS) y dan los valores de población de datos.

Sistema soporte para producción de programas (SMSS)
Habiéndose creado la biblioteca de segmentos genéricos de carga, y conocido el pedido del cliente (y por consiguiente sus datos) es ya el momento de ejecutar el sistema de producción (Fig. 4). Básicamente éste comprende dos procesos. El primero es el proceso de los DLS, el cual prepara la base de datos para cada elemento de control. Conocidos los valores de los datos por elemento de control, se puede automatizar la generación de los DLS. El segundo es el proceso por el que se construye una cinta magnética de carga para una central específica, la cual puede ser para una central nueva o para una ampliación.

Hay que hacer notar que el rigor y la fiabilidad son esenciales dentro de la producción; de nuevo, el contexto del ITT 1240 sirve para establecer el nexo entre el código del programa y el diseño de los datos a alto nivel.

Informe sobre el estado del CHILL

Durante el actual periodo de estudio se están considerando dos grandes ampliaciones del CHILL. La primera se refiere a la entrada/salida, y la segunda a la compilación por partes discretas y al manejo de programas muy grandes. Estos dos aspectos han consumido la mayor parte del tiempo de las reuniones. En ambas áreas se

ha alcanzado un considerable progreso. En la reunión de diciembre de 1982 se decidió adoptar el término "compilación por partes discretas" para cubrir tanto la compilación separada como la compilación independiente.

Entrada/salida
Ya existe un acuerdo sobre las características básicas a proveer. Se ha formulado un modelo del sistema de entrada/salida que delinea los límites entre las suposiciones hechas sobre el mundo exterior a los programas CHILL y las facilidades ofrecidas dentro de estos programas. Además, el modelo describe la interacción entre los tres estados de dicho sistema. El resto son detalles a resolver sobre la propuesta de entrada/salida y la definición clara de la sintaxis y semántica. Durante el periodo de estudio se intentará incluir facilidades especiales de entrada/salida de textos.

Compilación por partes discretas
Se ha alcanzado un acuerdo sobre un pequeño conjunto de conceptos generales para ampliar el CHILL. Estos conceptos tienden a hacer mínimos los impactos sobre el CHILL actual, sin dejar de ofrecer las facilidades esenciales necesarias para el manejo de grandes programas por compilación independiente segura, bibliotecas y gestión de espacio para nombres. La compilación por partes implica que se compila una parte fuera de su contexto final, o careciendo de una o más subpartes, combinándose luego las partes pre-compiladas. Dentro del CHILL, esto supone que se compilan partes con acceso a su contexto o disponiendo de una especificación del mismo. Al combinarse dichas partes se comprueba la consistencia de tal especificación con el contexto real para que la compilación sea segura e independiente. En este área el trabajo proseguirá con la especificación detallada de la propuesta y la completa definición de la semántica y sintaxis.

Conclusiones

Este artículo ha esbozado lo que es el CHILL y su estado actual en el correspondiente grupo de trabajo del CCITT. El entorno de programación del CHILL pertenece a un conjunto integrado de sistemas soporte utilizados para fabricar la central digital ITT 1240. A primera vista siempre pueden identificarse áreas de revisión y mejoras, pero la decisión de usar CHILL y la estrategia de programación se ven corroboradas por la instalación, con éxito, de centrales ITT 1240 en Bélgica y Alemania.

Referencia

1 D. A. Lawson: Desarrollo planificado de la programación ITT 1240 por diferentes centros: *Comunicaciones Eléctricas*, 1983, volumen 57, nº 4, págs. 284–288 (en este número).

Tani Haque se graduó en 1969 en la Universidad de Londres. Comenzó dedicándose a ingeniería eléctrica en Westinghouse y después en ICL, tras de lo cual pasó al campo de la programación. Ingresó en Standard Telecommunication Laboratories, donde dirigió el Centro de Control de Programación Básica (BSCC) que dio soporte al sistema METACONTA*10C; fue después designado jefe de programas soporte en el ESC de ITTE. Cuando se creó el PSC, el Sr. Haque pasó a esta nueva unidad, en la que ahora dirige la tecnología de programación y es director técnico delegado. Además representa en el CCITT a la casa líder de ITT en lenguajes SPC.

* Marca registrada del Sistema ITT

Biblioteca de proyecto

Los programas para sistemas de telecomunicación incrementan su complejidad a medida que los usuarios solicitan una gama más amplia de servicios. La documentación producida por ordenador mediante una biblioteca de proyecto es esencial para seguir la pista de los productos y subproductos creados durante el desarrollo. Sirve además de ayuda en la gestión del proyecto.

R. Raible
Standard Elektrik Lorenz AG, Stuttgart,
República Federal de Alemania

Introducción

Está ampliamente aceptado que una documentación íntegra de toda la información generada durante el desarrollo contribuye esencialmente a la calidad de los productos de programación. Dentro de la estructura de un proyecto, al mantenimiento de una correcta documentación debería, por consiguiente, concedérsele tanta importancia como al desarrollo de los programas. Esto significa que la documentación no debe ser considerada simplemente como un trabajo secundario sino, más bien, como una parte integral del desarrollo.

Se necesitan adecuadas herramientas de documentación que aseguren que todos los involucrados en un proyecto son conscientes de la importancia de la misma y la mantienen constantemente actualizada. Una primera etapa es la creación de una biblioteca de proyecto que facilite la organización y administración de todos los productos creados durante el desarrollo. En este contexto, un producto se define como un conjunto de documentos relativos a una sección, pequeña y bien definida, del proyecto total. Los productos se clasifican en grupos de productos del mismo tipo. Los tipos de producto corresponden a fases del proyecto, en tanto que definen, para un producto dado, qué documentos y programas específicos de la fase deben producirse, y dónde deben ser almacenados. Igualmente proporcionan a las personas dedicadas a un proyecto una guía para las actividades de desarrollo. Por otra parte, los tipos de producto ayudan a los directivos a planificar y controlar el proyecto.

Figura 1
Estructura global de la biblioteca de proyecto.

Modelo de proyecto

Todos los proyectos de programación se dividen en las siguientes fases:

- análisis
- definición
- diseño del sistema
- diseño de componentes
- realización de módulos
- realización de subsistemas
- realización de sistemas
- instalación
- operación y mantenimiento.

El modelo de proyecto ayuda al desarrollo, proporcionando descripciones detalladas de todas las fases, actividades, productos y medidas de gestión del proyecto, en segmentos claramente definidos que se procesan individualmente. Cada segmento tiene funciones y resultados bien definidos.

Estructura de la biblioteca de proyecto

La biblioteca de proyecto consta de tres partes, según se indica en la figura 1. En primer lugar, un *banco de productos* para registrar todos los productos y sus subproductos obtenidos durante el desarrollo. En segundo lugar, la *administración del producto,* que consta de los ficheros de catálogo del producto, de un fichero de la *jerarquía del producto,* y de ficheros modelo (ficheros maestros para guías (proforma)

de subproducto, informes de revisión y fallos, etc.). Finalmente tenemos la parte de *gestion del proyecto,* que comprende la información de gestión para control y planificación del proyecto, así como la historia del mismo.

Banco de productos

La estructura del banco de productos asegura que, a través de todo el desarrollo, los ingenieros del proyecto sepan claramente dónde deben almacenar y obtener los elementos de información específicos. Para conseguir esto se utiliza la estructura tridimensional mostrada en la figura 2.

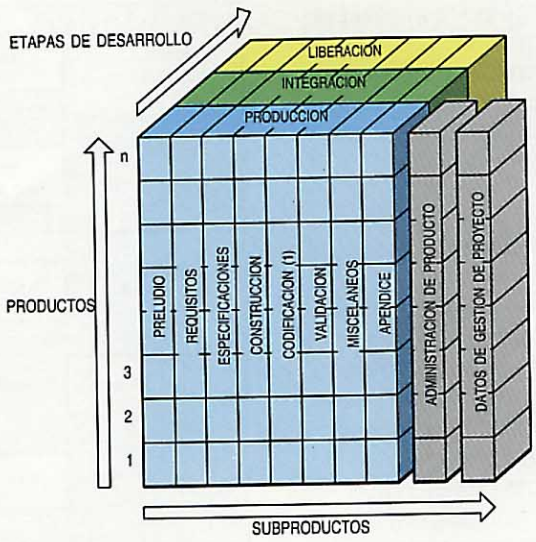


Figura 2
Banco de productos tridimensional.

Productos
Como criterio de ordenación de los productos en el *banco de productos* se adopta una estructura de sistema. Dado que esta estructura cambia en el curso del proyecto, según se intercalan o reclasifican productos, la actualización constante del sistema de ordenación implicaría gastos considerables. Por consiguiente, esta dimensión se considera como un gran grupo de productos que cambia según progresa el proyecto, pero en el que a cada producto se le asigna un nombre único. Los criterios lógicos de ordenación son definidos por la administración asociada al producto.

Subproductos
La segunda dimensión contiene los subproductos relacionados con un producto dado. Los nombres, números, y contenidos de los subproductos individuales están determinados por cinco tipos de producto: *sistema, componente, módulo, subsistema* y *sistema integrado.* Su interdependencia se muestra en la figura 3.

La estructura de los subproductos se define mediante guías (proforma) específicas del tipo de producto. Las guías del subproducto son modelos estrictamente definidos, concebidos como instrucciones de trabajo para la producción de los subproductos individuales.

Como ejemplo, los subproductos correspondientes al tipo *módulo* podrían ser:

- apéndice (= app)
- parte de código *1* (= c01)
- construcción (= con)
- misceláneos (= mis)
- prestaciones (= per)
- especificaciones (= spe)
- controlador de pruebas *1* (= td1)
- validación (= val).

Etapas de desarrollo
La tercera dimensión proporciona la base para el control de calidad y planificación del proyecto. Pueden distinguirse tres etapas de desarrollo del producto: producción, integración, liberación. Un subproducto puede estar en cualquiera de ellas en momentos específicos.

El paso de un subproducto desde una etapa a otra se controla mediante grupos de revisión e inspección que usan procedimientos normalizados. Se asignan estados de desarrollo a las correspondientes etapas para conseguir una descripción exacta de los procedimientos normalizados. Estos estados definen los hitos para los subproductos dentro de una etapa de desarrollo.

En la etapa de producción, un subproducto se encuentra en uno de estos cinco estados: *definido, en producción, en pruebas, en corrección,* o *en aceptación.* Esta etapa corresponde al desarrollo de la primera versión de un subproducto o a la corrección de una ya existente. Puede utilizarse también para planificar productos y sus subproductos, crear subproductos adicionales y revisar subproductos especificados.

Un subproducto en la etapa de integración se encuentra en el estado *aceptado;* se

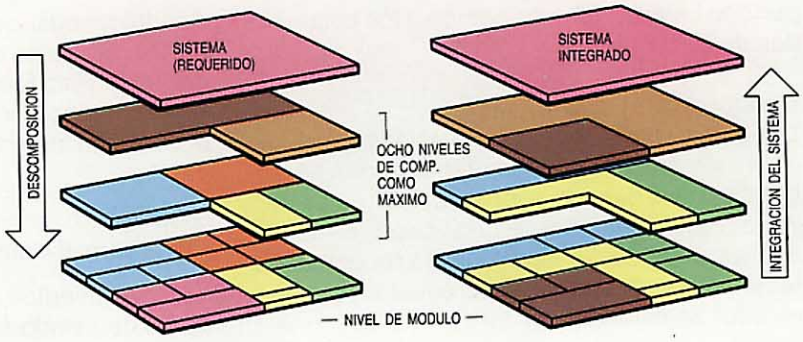


Figura 3
Interdependencia entre tipos de producto.

asigna al miembro del equipo responsable de las pruebas de integración.

A la etapa de liberación le corresponde el estado *liberado,* que se asigna al miembro del equipo encargado de las pruebas de aceptación.

Administración de producto

Los datos del catálogo de producto y los datos de la guía (proforma) se mantienen en la biblioteca del proyecto con el fin de que, en cualquier momento, pueda obtenerse información sobre los productos del banco, junto con sus relaciones internas.

El catálogo de productos es el índice central para todos los productos del banco. Contiene una declaración para cada producto. Se utiliza para:

- identificar un producto y su tipo
- describir el estado de desarrollo actual de cada subproducto
- controlar los números de versión de las versiones liberadas y en integración
- controlar el lenguaje de programación de los subproductos que contienen codificación.

Las guías (proforma) de subproducto se diseñan para ayudar a los usuarios de una biblioteca a realizar sus tareas eficientemente. Pueden ser normas que describan el contenido y organización de documentos y programas, o impresos formatados para registrar las decisiones de diseño. Alternativamente pueden ser sugerencias para revisiones planificadas, programas maestros para prueba de programas de usuario (controladores de pruebas), o información auxiliar.

Supervisión de proyecto

El ciclo de vida de un producto se describe mediante los estados de desarrollo. La definición de procedimientos normalizados para el cambio desde un estado a otro proporciona un punto de referencia para el control del progreso del proyecto. Resulta posible planificar el proyecto asignando períodos específicos de tiempo a los estados de desarrollo.

Control de proyecto

La figura 4 es un diagrama de estados que ilustra los procedimientos normalizados de transferencia para un subproducto.

Se usan los estados especificados en la figura para describir los procesos recientes de desarrollo de un subproducto. Tales estados se caracterizan así:

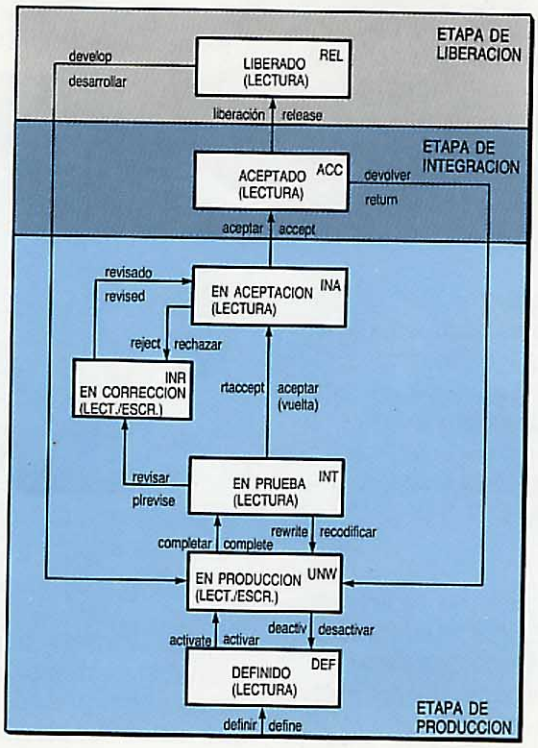


Figura 4
Ciclo de vida de subproducto.

- *Definido:* cuando el subproducto ha sido creado en la biblioteca del proyecto.
- *En producción:* cuando se ha creado una versión de trabajo del subproducto, ha sido designado un miembro del equipo para realizar la documentación y codificación, y se han efectuado modificaciones.
- *En prueba:* cuando el subproducto se ha completado según el criterio del responsable y van a realizarse medidas de control de calidad (revisiones, etc.).
- *En corrección:* cuando se han generado nuevas ediciones para modificación del subproducto como consecuencia de las medidas de control de calidad, pero sin revisión adicional.
- *En aceptación:* cuando el administrador del proyecto comprueba que se están incorporando los cambios resultantes del control de calidad, y que el subproducto ha sido verificado en cuanto al cumplimiento de requisitos normalizados (p. ej.: estructura de documentos, reglas de codificación, integridad de la documentación).
- *Aceptado:* cuando el subproducto cumple los requisitos de las normas desde un punto de vista formal y puede utilizarse, junto con otros programas, para las pruebas de integración.
- *Liberado:* un subproducto está liberado cuando se ha entregado una versión válida para el cliente.

Los procedimientos normalizados para los cambios de estado se apoyan en órdenes

de la biblioteca de proyecto. Seguidamente se describen las principales funciones de estos procedimientos.

Intercambio de información. Un cierto número de órdenes de la biblioteca de proyecto están diseñadas para el intercambio de información entre los usuarios de la biblioteca.

Procedimiento de definición. Con él comienza el desarrollo del producto, de lo cual se informa a la biblioteca del proyecto. Se caracteriza así: la definición depende del tipo de producto, se crea una versión de partida para cada subproducto determinado por dicho tipo de producto, y se inicializa por medio de las guías de subproducto.

Procedimiento de revisión. En términos formales, todos los subproductos pasan por todos los estados de desarrollo descritos en la figura 4. Sin embargo, existen diferencias entre los subproductos individuales, según su impacto sobre el control de calidad. Para algunos subproductos, es suficiente una lectura de código por parte de otro miembro del equipo. En otros casos (p. ej.: especificaciones, requisitos) es necesaria una reunión de revisión.

Procedimiento de modificación. Las necesidades de modificación (p. ej.: eliminación de las faltas detectadas en subproductos aceptados y, posiblemente también, liberados) pueden obligar a devolver los correspondientes productos a la etapa de producción. A ella se transfiere una copia del subproducto a modificar, realizándose allí los cambios. En el banco de producto se registran todas las modificaciones sobre el subproducto, de tal forma que, en todo momento, pueda obtenerse cualquier versión anteriormente existente.

Historia del proyecto

En el fichero histórico del proyecto se almacenan, en orden cronológico, los cambios de estado y los principales eventos en el ciclo de vida de los productos. Dentro de la estructura de control de versiones, la producción de cada nueva versión requiere una breve explicación acerca de su necesidad (p. ej.: requisitos de modificación en la nota de revisión).

Planificación de proyecto

La planificación de proyecto se basa en la asignación de información de gestión a los productos y subproductos que van a desarrollarse. Pueden conservarse dos tipos de datos de planificación: datos reales y datos planificados. Los datos reales se almacenan automáticamente cuando se llama a las órdenes de la biblioteca de proyecto, mientras que los datos planificados están asigna-

Tabla 1 — Ordenes de biblioteca de proyecto

Cambios de estado	
accept	cambio desde estado *ina* a estado *acc*
activate	cambio desde estado *def* a estado *unw*
complete	cambio desde estado *unw* a estado *int*
deactiv	cambio desde estado *unw* a estado *def*
develop	cambio desde estado *rel* a estado *unw*
plrevise	cambio desde estado *int* a estado *inr*
reject	cambio desde estado *ina* a estado *inr*
release	cambio desde estado *acc* a estado *rel*
return	cambio desde estado *acc* a estado *unw*
revised	cambio desde estado *inr* a estado *ina*
rewrite	cambio desde estado *int* a estado *unw*
rtaccept	cambio desde estado *int* a estado *ina*
Administración de producto	
catinfo	editar información de catálogo de producto
catmod	modificar información de catálogo de producto
dearchive	redefinir un producto suprimido en la biblioteca de proyecto
define	definir un nuevo producto
defsp	definir un nuevo subproducto para un producto existente
deletesp	eliminar un subproducto de un producto
pldelete	eliminar un producto de la biblioteca de proyecto
plfetch	copiar un subproducto desde la biblioteca de proyecto para modificación
plmod	devolver un subproducto modificado a la biblioteca de proyecto
plprint	editar información de subproducto
plread	copiar una versión de un subproducto desde la biblioteca de proyecto
plrename	cambiar el nombre de un producto
projstat	generar resumen de estados de proyecto
retrieve	editar una lista de productos con unos atributos dados
scat	editar información resumida de catálogo de producto
schema	copiar archivos (proforma)
structure	editar lista estructurada de productos
version	editar información histórica de subproductos
Supervisión de proyecto	
deadline	editar estado de productos retrasados
history	editar información de la historia del proyecto
mgmtinfo	editar información de gestión
mgmtmod	modificar información de gestión
projover	generar resúmenes del proyecto
projplan	generar tablas de planificación del proyecto
revdate	editar lista de fechas de revisión
schedule	generar diagramas de barras GANTT
trend	generar resúmenes de tendencias de planificación
Gestor de órdenes	
change	cambiar biblioteca de proyecto en uso
exit	sacar el gestor de órdenes de la biblioteca de proyecto
plwork	introducir el gestor de órdenes de la biblioteca de proyecto
Misceláneos	
abb	editar abreviaturas
integ	verificar la integridad de una biblioteca de proyecto
memoread	copiar notas
plhelp	editar información sobre órdenes de la biblioteca de proyecto
plinstall	instalar una nueva biblioteca de proyecto

dos a productos y subproductos. Para cada subproducto se almacenan fechas de comienzo, revisión y finalización; en cambio los esfuerzos y tamaños se almacenan para el conjunto del producto. Esta base de datos se conserva en la parte de gestión de la biblioteca del proyecto.

Ordenes de la biblioteca de proyecto

Las órdenes facilitan el trabajo con la biblioteca y aseguran la consistencia de los datos almacenados dentro de la misma. La tabla 1 muestra las órdenes disponibles. Cada una de las cuatro clases de usuario siguientes puede utilizar un subconjunto de todas las órdenes.

- Los administradores de proyecto son responsables de la biblioteca y del mantenimiento y registro de todos los datos en la misma.
- Los planificadores de proyecto pueden obtener toda la información necesaria sobre planificación, pero no tienen responsabilidad sobre ningún producto.
- Los planificadores de producto son responsables del desarrollo del mismo; pueden modificar los datos de planificación para sus productos y cambiar los estados de desarrollo.
- El personal de desarrollo tiene acceso de escritura a productos y subproductos en los estados de *producción* y *corrección*.

Conclusiones

La documentación producida por ordenador mediante una biblioteca de proyecto, proporciona a las personas relacionadas con un producto una estructura para sus actividades de desarrollo, especificando los productos, su contenido y el lugar donde deben ser archivados. Los contenidos pueden seleccionarse y obtenerse de acuerdo con diversos criterios. Por otra parte, dicha biblioteca mantiene información actualizada del estado del proyecto, ofreciendo almacenamiento y recogida centralizados para todos los datos de producto y subproducto obtenidos durante el proyecto. Finalmente, la biblioteca de proyecto facilita la planificación y control mediante secuencias normalizadas y procedimientos de entrega para los subproductos. Una biblioteca tal fue desarrollada en Standard Elektrik Lorenz para el proyecto Bildschirmtext en cooperación con Softlab, una firma de programación de Munich.

La biblioteca de proyecto se emplea actualmente en diversos desarrollos de programación y equipo físico que ocupan de tres a sesenta personas. Se ha realizado en un VAX 11/780 con sistema operativo VMS y una pequeña parte del IS/1. Los buenos resultados obtenidos hasta la fecha con el sistema han estimulado el uso de este concepto de biblioteca como herramienta normalizada para futuros proyectos de programación.

Roland Raible nació en 1952 en Ludwigsburg, Alemania Occidental. Estudió matemáticas e informática en la Universidad de Stuttgart, donde obtuvo el grado de Diplomado en Matemáticas. En 1977 ingresó en SEL para trabajar en el desarrollo de un simulador de entorno para prueba de los programas 12R. Dos años más tarde pasó al grupo de programas soporte del Bildschirmtext, trabajando en la biblioteca de proyecto desde mediados de 1980.

Modificación dinámica de programas

En muchas industrias, el equipo controlado por ordenador debe funcionar sin cesar, aun cuando se requiera una modificación o reparación. Para ello, los programas del ordenador deben, igual que los módulos de equipo, poderse reemplazar sin afectar a la operación del sistema.

J. A. De Man
M. A. E. van Rumste
Bell Telephone Manufacturing Company, Amberes, Bélgica

Introducción

Los sistemas de control por ordenador juegan un papel cada vez más importante en la sociedad moderna. Algunos son tan vitales que exigen un funcionamiento continuo. Entre ellos los sistemas de conmutación telefónica, los de control de procesos que supervisan grandes plantas industriales, los de reserva de plazas y control de tráfico en las líneas aéreas.

Tales sistemas de control suelen construirse a base de componentes electrónicos, incluyendo microprocesadores y su lógica asociada, cuyos programas van de lo simple a lo complejo. Cuando se requiere funcionamiento continuo, los módulos del sistema tanto de equipo como de programas deben ser capaces de admitir faltas y reparaciones con el sistema funcionando, sin afectar a su operación.

Son bien conocidas las técnicas para hacer que el equipo admita fallos (p. ej., memoria con lógica de detección y corrección de errores, sistemas de doble procesador), y aquí no se consideran.

Con frecuencia los programas son complejos y están sujetos a cambios por dos razones. Primero, parece imposible desarrollar sistemas de programación exentos de errores (suelen ser más complejos en varios órdenes de magnitud que los sistemas de equipo y no hay todavía metodologías y herramientas de programación adecuadas), y su corrección requiere cambios en los programas. Segundo, la funcionalidad mejora continuamente durante la vida del programa. Ya que los sistemas-programa son flexibles, se les hacen mejoras a menudo.

Considerando la analogía con el equipo es razonable admitir que los cambios en programas se realicen mediante la sustitución de módulos, siempre que el sistema-programa esté construido modularmente. Tal sustitución, sin interrumpir el funcionamiento del sistema, se denomina *modificación dinámica de programas*. En este artículo se exponen las condiciones necesarias para conseguirlo.

Módulos-programa

Aunque sea difícil de probar, la práctica general indica que una estructura modular ayuda al desarrollo de programas de buena calidad, mantenibles a un coste razonable. Es también necesaria para poder mantener los programas mientras opera el sistema, sustituyendo módulos. Lo primero es saber qué se entiende por módulo-programa.

Los programadores dividen los programas extensos y complejos en partes más pequeñas, por razones tales como:

- Los programas grandes son difíciles de comprender en conjunto. Dividiéndolos en bloques más pequeños y sencillos, puede llegarse a partes comprensibles para un solo programador.
- Estas partes pueden reutilizarlas los programadores sin necesidad de copiar la misma codificación en los diferentes sitios donde se necesite.
- Dividir el programa en porciones es esencial, si trabajan distintas personas con el mismo programa.

Usando módulos-programa, el sistema-programa (Fig. 1), con su compleja funcionalidad (esto es, lo que hace) y su interfaz con el entorno (cómo se activan y ejecutan sus funciones en dicho entorno), se divide en bloques constructivos (Fig. 2) con tres propiedades principales: funcionalidad menos compleja que la del sistema total, limitado interfaz con el entorno, y relación mediante interfaces entre módulos con las otras partes del sistema.

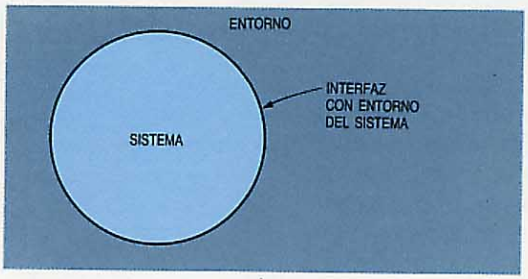


Figura 1
Sistema-programa y sus interfaces con el entorno.

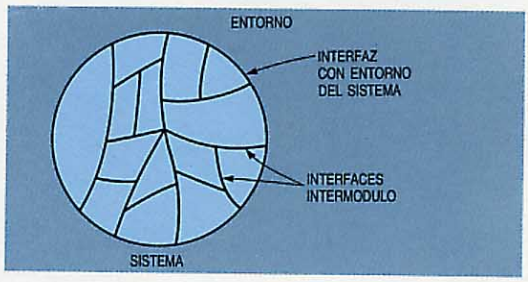


Figura 2
Sistema-programa construido como colección de módulos.

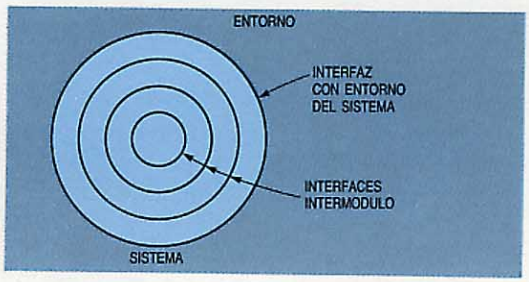


Figura 3
Sistema estructurado en niveles.

La división en módulos del sistema-programa se debe hacer con mucho cuidado. Debería reducirse al mínimo tanto el número de módulos que se relacionan con el entorno, como el de mecanismos para interfaz entre módulos, siendo estos mecanismos sencillos y directos. La partición la debería hacer un equipo de expertos diseñadores de sistemas.

La división indicada en la figura 2 seguramente no es la más elegante ni la más eficaz. Como se puede ver, casi todos los módulos poseen interfaces con el entorno, y todos ellos deben tener la funcionalidad necesaria para relacionarse entre sí. Tal sistema podría ser muy complicado. Un sistema estructurado en niveles, como se ilustra en la figura 3, se acomoda mejor a las necesidades actuales, ya que cada módulo tiene muy pocos interfaces, y sólo un módulo se relaciona con el entorno.

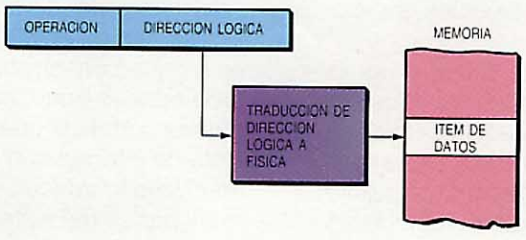


Figura 4
Referencias a ítems de datos.

Las figuras 2 y 3 indican que los módulos-programa son bloques constructivos que funcionan juntos, tratando adecuadamente el interfaz del sistema con el entorno, y consiguiendo la deseada funcionalidad del sistema. En tal contexto, un módulo-programa tiene dos importantes propiedades: un *interfaz* bien definido, visto desde otros módulos, y una *funcionalidad* perfectamente delimitada (*lo que* el módulo hace).

Dentro del ámbito de modificación dinámica de programas, un determinado módulo sólo deberá ser reemplazable por otro con al menos igual interfaz y funcionalidad.

Un módulo puede realizarse en diferentes formas: estructura de datos, subprograma (función, subrutina o procedimiento), o incluso programa completo que se ejecuta en concurrencia con otros programas del sistema. Se pueden considerar varias formas de interfaz entre módulos:

- Las variables individuales de una estructura de datos, como un bloque COMUN, pueden ser direccionadas por módulos que tengan una referencia a la estructura de datos. Esta forma de interfaz, que se denomina *acceso directo,* la ofrecen muchos lenguajes de programación.
- Un subprograma se puede activar si el módulo que llama conoce convenios de llamada (es decir, nombre del subprograma y tipos de datos parametrizados). Esto se denomina *interfaz de procedimiento.* El soporte necesario para la activación de subprogramas se encuentra en todos los lenguajes de programación modernos.
- El interfaz entre diferentes programas de ejecución concurrente suele ser *controlado por mensajes.* Muy pocos lenguajes de programación dan soporte al intercambio de mensajes entre diferentes módulos-programa; entre ellos figuran el C (con llamadas a las rutinas del sistema operativo), el CHILL y el ADA. La arquitectura INTEL iAPX432 ofrece soporte físico para construir sistemas con interfaz por mensajes[1].

Debe analizarse la forma en que se realizan estos interfaces en los procesadores típicos, ya que ello influye sobre la sustitución de los módulos-programa existentes por nuevos módulos.

El *acceso directo* a las variables de la estructura de datos se incorpora siempre mediante especificación de la dirección lógica del dato referenciado en el campo de la instrucción (Fig. 4). El procesador ofrece luego un mecanismo para traducir esta dirección lógica en dirección física. Tales esquemas de traducción varían desde los

sencillos (por ejemplo, dirección lógicas igual a dirección física) hasta los basados en la capacidad, que ofrecen información y protección centralizada de la dirección física, pasando por el direccionamiento de una tabla de páginas y de segmentos (un segmento es una parte de memoria física de longitud variable, mientras que una página tiene longitud fija). La figura 5 indica las estructuras básicas de datos necesarias para un direccionamiento por capacidad. Una referencia de datos o dirección lógica en una instrucción consta de un selector de capacidad (que selecciona una de las capacidades en la lista del módulo-programa) y una dirección relativa, apuntando al dato particular del segmento de memoria. La capacidad mantiene un selector de segmento, indicando el descriptor del segmento en la tabla de segmentos del sistema. Usando la dirección de base física y la longitud, el descriptor referencia al segmento en memoria. La capacidad también contiene los derechos de capacidad (una indicación del tipo de operaciones que el módulo-programa puede realizar en el segmento), mientras que el descriptor incluye el tipo de segmento. Cualquier conflicto entre direccionamiento relativo y longitud del segmento, o entre derechos de capacidad y tipo de segmento, es detectado por la arquitectura, ofreciendo este mecanismo protección total.

El *interfaz de procedimientos* usa una lista de direcciones lógicas, parametrizadas, para pasar los parámetros reales del procedimiento, y la instrucción *salto a subrutina* o *llamada* consigue el cambio necesario en el flujo de control del programa. La dirección de destino de la llamada se especifica también en el formato lógico de direcciones del procesador.

En los procesadores clásicos, el *interfaz por mensajes* se realiza mediante interfaces de procedimientos con las rutinas del sistema operativo. Valen las observaciones hechas sobre direcciones lógicas y físicas.

Otra propiedad importante de los módulos-programa es que al menos existen dos tipos; los que tienen y los que no tienen *información de estado interno.* Esta información se almacena en los datos internos del módulo. Cuando se crea éste, los datos internos se inician con ciertos valores, que luego cambian durante la vida del módulo.. La historia de cada variable interna es parte de la historia completa del módulo. La figura 6 indica este efecto en un módulo compuesto de un segmento de datos y un segmento de código. Durante la vida del módulo, el código permanece sin cambios y los datos probablemente se alteren a cada nueva referencia al módulo.

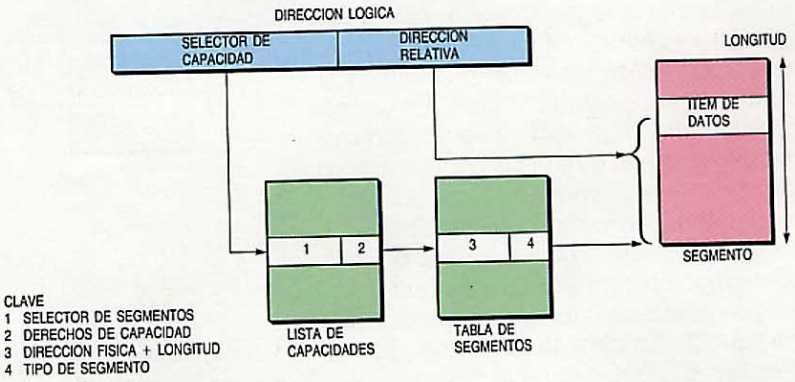


Figura 5
Direccionamiento basado en la capacidad.

Sustitución dinámica de módulos-programa

El siguiente problema será considerar cómo se sustituye un módulo-programa con el sistema en funcionamiento. Como se ha dicho, un módulo sólo puede ser reemplazado por otro que al menos tenga el mismo interfaz y funcionalidad. Para que un módulo-programa pueda sustituirse de modo dinámico hay que disponer de dos importantes mecanismos.

Primero: se necesita conmutar del módulo existente al nuevo en el momento elegido. Esto puede ser difícil si cada vez que curse una referencia al módulo (dirección lógica) hay que actualizarla. Por lo tanto, el mecanismo de direccionamiento del procesador debería ofrecer una fuerte separación entre dirección lógica y física; la información de la última debería estar

Figura 6
Cambios del estado interno de un módulo-programa.

centralizada para todo el sistema, generalmente en tablas de segmentos, donde cada una tiene sólo una entrada. En tales condiciones, la dirección lógica permanece igual, y sólo habrá que actualizar la entrada a la tabla de segmentos. Un mecanismo de direccionamiento por capacidad ofrece esta posibilidad sin dificultades.

Segundo: para sustituir un módulo-programa con información del estado interno, se requerirá un método para transferir esa información a la nueva versión. Como esta transferencia puede consumir un tiempo considerable, mientras se esté ejecutando el antiguo módulo debe permanecer activo.

Hay un proceso típico para la sustitución dinámica de un módulo-programa (Fig. 7)[2]. Consideremos que se utiliza una tabla de segmentos con una sola entrada para dicho módulo. Todas las referencias al módulo se tratan por medio de esa entrada. Inicialmen-

te existe una sola versión (Fig. 7a), la cual ejecuta las operaciones pedidas que provocan el cambio de las variables internas, y modifica así su estado interno. En un momento dado, un programa de utilidad (normalmente es un gestor de configuración) carga una nueva versión del módulo en la memoria del sistema (Fig. 7b). Una vez completado este proceso, el gestor de configuración comienza a transferir los antiguos estados internos a la versión ahora introducida. Durante la transferencia (Fig. 7c) el módulo antiguo permanece activo. Completada esta fase, el gestor de configuración destruye toda referencia al módulo antiguo y permite el acceso al nuevo. En un sistema con tabla de segmentos centralizada, esta conmutación se puede ejecutar sin más que modificar la entrada única para el módulo (Fig. 7d).

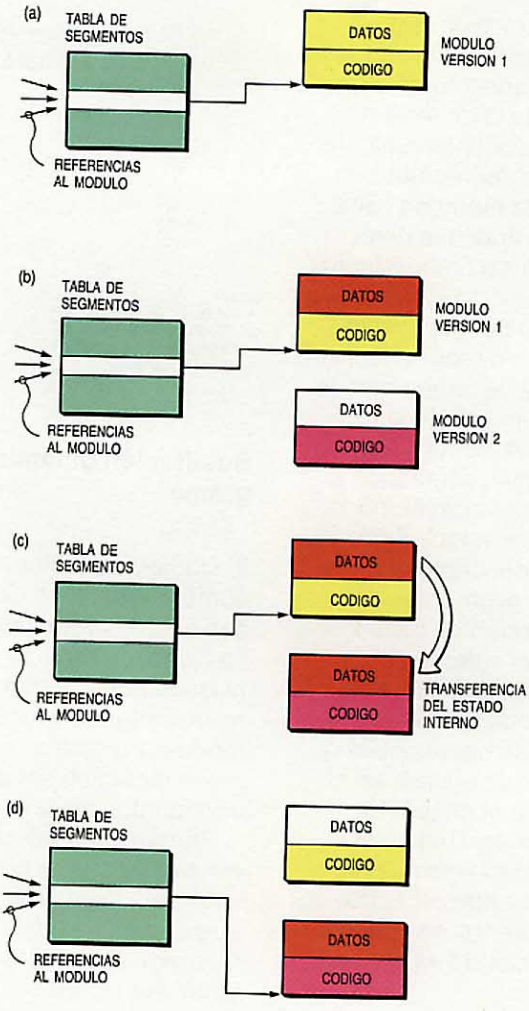


Figura 7
Sustitución de un módulo-programa:
a) operación normal del sistema
b) cargada la nueva versión del módulo
c) transferencia del estado del módulo
d) operación normal del sistema después de la sustitución.

Tipo de dato abstracto

De la anterior descripción informal surgen dos importantes preguntas:

- ¿Cómo describir con exactitud el interfaz y la funcionalidad de un módulo?
- ¿Cómo comprobar, en tiempo de desarrollo y de ejecución del programa, que una nueva versión de un módulo tenga el mismo interfaz y funcionalidad que la original?

Las respuestas en parte pueden hallarse en el concepto teórico de *tipo de dato abstracto (axiomático)* y en el uso práctico de *lenguajes de programación de alto nivel.*

El *tipo de dato abstracto* debe considerarse extensión natural de los tipos de datos, tal como se definen en los lenguajes de programación. Un objeto (o variable) de un cierto tipo posee dos propiedades importantes: durante su vida recibe *valores* incluidos en un determinado conjunto, y sólo pueden utilizarlo *operaciones* bien definidas que le harán cambiar de valor.

Muchos lenguajes de programación ofrecen tipos de datos definidos en el lenguaje. La figura 8 muestra cómo se representa en este artículo un objeto de cierto tipo de datos; concretamente un objeto entero. Conforme a la idea de tipo de dato, un buen lenguaje de programación debería limitar las operaciones sobre un objeto a aquéllas que el propio tipo de objeto defina. Además del consistente uso de tipos de datos definidos en el lenguaje, muchos lenguajes permiten también construir tipos de datos definidos por el usuario, tomando los primeros como bloques constructivos. Mediante tal facilidad, puede, por ejemplo, definirse un tipo de dato de números complejos utilizando como componentes números reales:

```
tipo COMPLEJO es
    registro
        RE: REAL;                    (1)
        IM: REAL;
    fin de registro;
```

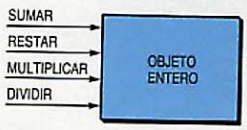


Figura 8
Representación de un objeto por sus operaciones.

Sin embargo, esta definición no revela nada sobre las operaciones aplicables a objetos del tipo *complejo,* que pueden definirse usando otras construcciones del lenguaje, como los subprogramas (procedimientos y funciones). Para los números complejos tienen significado las operaciones mostradas en (2), las cuales definen funciones para crear, sumar, restar, multiplicar y dividir números complejos, mediante operaciones válidas para números reales.

```
función CREAR-COMPLEJO   (X, Y: REAL) regresar COMPLEJO;   (2)
función "+"              (X, Y: COMPLEJO) regresar COMPLEJO;
función "−"              (X, Y: COMPLEJO) regresar COMPLEJO;
función "*"              (X, Y: COMPLEJO) regresar COMPLEJO;
función "/"              (X, Y: COMPLEJO) regresar COMPLEJO;
```

Un objeto del tipo *complejo* se puede representar como en la figura 9. Las definiciones en (1) y (2) dan descripciones alternativas de dicho tipo. La primera detalla la representación (cómo se compone la estructura de datos a base de estructuras más simples), mientras la última describe las operaciones realizables sobre objetos de ese tipo. Matemáticamente, los detalles de la representación son irrelevantes, y se prefiere la definición funcional. Es, también, más directa para módulos-programa sustituibles. Se llama definición de *tipo de dato abstracto* o axiomático a la que especifica las operaciones relevantes e ignora la estructura de datos subyacente.

Los tipos de dato abstractos son una clase de módulo-programa, con funcionalidad e interfaz bien definidos. Están matemáticamente justificados y forman excelentes bloques constructivos para sistemas-programa complicados.

Un lenguaje de programación de tecnología actual, como el Ada, ha de acomodarse a los tipos de datos abstractos. Los *paquetes* Ada se pueden usar para introducir módulos-programa en general, y tipos abstractos de datos en particular. Un *paquete* es la unidad básica para definir una colección de entidades relacionadas lógicamente, tales como una zona común de datos y tipos, una colección de subprogramas relacionados o un conjunto de declaraciones y operaciones asociadas al tipo.

Correspondiendo al interfaz y funcionalidad del módulo-programa, un paquete Ada consta de dos partes: especificación del *interfaz* (constantes públicas, tipos, variables, subprogramas y convenios de llamada del subprograma) y *cuerpo* o *realización* (modo en que el paquete incorpora la funcionalidad deseada). En tiempo de desarrollo del programa, el compilador Ada comprueba que la realización casa con el interfaz, y que los otros módulos con los que se relaciona usan dicho interfaz correctamente.

Como ejemplo, con un paquete Ada se puede construir un módulo-programa que defina la aritmética de los números complejos, y que podría insertarse en una biblioteca de subprogramas matemáticos. Esta fue una de las razones que apoyaron el tratamiento formal de módulos en Ada. A partir de las definiciones de la estructura de datos en (1) y de las operaciones en (2), se obtendría un paquete como el (3), compuesto de especificación y cuerpo; sin embargo, la definición (3) tiene todavía una desventaja. La estructura de un objeto del tipo *complejo* se declara como un tipo de paquete público, y por lo tanto la conoce todo programa que se refiera a este paquete. Para ajustarse a la idea de tipo de dato abstracto, este detalle de representación debería estar oculto. Con el lenguaje Ada hay dos maneras de ocultar la información y construir tipos de datos abstractos; así pueden utilizarse *paquetes* Ada y *tipos privados* Ada para constituir *paquetes transformadores* o *gestores de tipos.* El gestor de tipos sólo puede actualizar la versión del tipo de dato abstracto que le es suministrada a través de parámetros forma-

```
paquete NUMEROS-COMPLEJOS es                                        (3)
  tipo de COMPLEJO es
    registro
      RE: REAL;
      IM: REAL;
    fin de registro;
  función CREAR-COMPLEJO  (X, Y: REAL) regresar COMPLEJO;
  función "+"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "–"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "*"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "/"             (X, Y: COMPLEJO) regresar COMPLEJO;
fin NUMEROS-COMPLEJOS;

cuerpo paquete NUMEROS-COMPLEJOS es
  función CREAR-COMPLEJO (X, Y: REAL) regresar COMPLEJO es
    Z: COMPLEJO;
  comienzo
    Z.RE = X;
    Z.IM = Y;
    regresa Z;
  fin CREAR-COMPLEJO;

  función "+" (X, Y: COMPLEJO) regresar COMPLEJO es
    Z: COMPLEJO;
  comienzo
    Z.RE = X.RE + Y.RE;
    Z.IM = X.IM + Y.IM;
    regresa Z;
  fin "+";

  función "–"     (X, Y: COMPLEJO) regresar COMPLEJO es ... fin;
  función "*"     (X, Y: COMPLEJO) regresar COMPLEJO es ... fin;
  función "/"     (X, Y: COMPLEJO) regresar COMPLEJO es ... fin;
fin NUMEROS-COMPLEJOS;
```

Figura 9
Representación de un objeto del tipo *complejo*.

```
paquete NUMEROS-COMPLEJOS es                                        (4)
  tipo COMPLEJO es privado;
    – La representación de objetos del tipo COMPLEJO
    – está oculta al usuario. Sólo pueden hacerse
    – asignación, comparación para igualdad y
    – desigualdad, y las operaciones que siguen, con
    – objetos del tipo COMPLEJO
  función CREAR-COMPLEJO  (X, Y: REAL) regresar COMPLEJO;
  función "+"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "–"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "*"             (X, Y: COMPLEJO) regresar COMPLEJO;
  función "/"             (X, Y: COMPLEJO) regresar COMPLEJO;
privado
  tipo COMPLEJO es
    registro
      RE: REAL;
      IM: REAL;
    fin registro;
fin NUMEROS-COMPLEJOS;
```

les. El tipo de dato *complejo* se podría describir mejor mediante el gestor de tipo de (4), el cual maneja todos los objetos del tipo *complejo*. Además, el usuario no tiene información sobre la estructura real de datos que representa a un objeto de este tipo. El sistema soporte de la programación Ada garantiza esta propiedad.

Quizá no se necesite una tajante separación entre los detalles representativos y una visión operacional en un tipo tan simple como los números complejos. El ejemplo de una cola de espera es más específico. Se puede construir un gestor de tipos para tratar colas de espera en un sistema operativo, como se muestra en (5).

```
paquete GESTOR-COLA-DE-ESPERA es                    (5)
  tipo COLA-DE-ESPERA es privado;
  – La representación de objetos del tipo COLA-
  – DE-ESPERA queda oculta al usuario. Sólo
  – pueden hacerse asignación, comparación de
  – igualdad y desigualdad y las operaciones que
  – siguen, con objetos del tipo COLA-DE-ESPERA.
  función CREAR regresar COLA-DE-ESPERA;
  procedimiento MECOLA-DE-ESPERA (X: ITEM;
  C: COLA-DE-ESPERA);
  procedimiento SACOLA-DE-ESPERA (X: fuera
  ITEM; C: COLA-DE-ESPERA);
privado:
  – En la incorporación del objeto COLA-DE-ESPERA,
  – usamos una representación ordenada de objetos
  – del tipo ITEM. También se podría imaginar una
  – representación como lista de objetos encade-
  – nados, pero esta elección no le afecta al usuario
  – del gestor de tipos. Punto débil del Ada es que
  – su información privada aparezca textualmente
  – en la especificación del paquete. Sin embargo,
  – ello se consideró útil para facilitar la formación
  – del compilador.

  LONG-MAX: constante = 100;

  tipo REP-COLA-DE-ESPERA es
    registro
      VALOR:          ordenación (1..LONG-MAX)
                      del ITEM
      LONG:           entero: = 0;
      CABEZA, COLA: entero: = 1;
    fin de registro;
  tipo COLA-DE-ESPERA es acceso REP-COLA-
  DE-ESPERA;
fin GESTOR-COLA-DE-ESPERA;
```

La realización como gestor de tipos es muy útil desde el punto de vista de la abstracción. Debe advertirse que los gestores de tipos *no* tienen información del estado interno, ya que ésta se guarda dentro de cada objeto tratado por el gestor. Ejecutando una operación del gestor, cambiará el estado interno del objeto, que le fue pasado como parámetro durante la llamada de operación. Con vistas a poder sustituir un módulo-programa durante la ejecución del programa, hay que atender a dos aspectos: sustitución del gestor de tipos y cambio de representación de *todos* los objetos creados y controlados por dicho gestor.

La segunda tarea no es trivial. Los objetos de cierto tipo quizá estén dispersos por todo el sistema-programa y su vida puede estar unida a otras partes del sistema, variando desde la existencia durante la activación de un subprograma hasta la vida entera del sistema.

Este problema se puede evitar usando una *estructura encapsulada de datos,* la cual define operaciones que manipulan solamente las variables internas del módulo. En contraste con un gestor de tipo, esta estructura tiene información de estado interno, almacenada en sus propias variables, la cual permanece oculta al usuario. En Ada, una realización de cola de espera se puede definir como una estructura de dato encapsulada mediante la siguiente declaración de paquete:

```
paquete COLA-DE-ESPERA es                           (6)
  procedimiento MECOLA-DE-ESPERA (X: ITEM);
  procedimiento SACOLA-DE-ESPERA (X: fuera ITEM);
fin COLA-DE-ESPERA;
```

Este paquete proporciona solamente un objeto (una versión) del tipo de dato deseado. Además, este objeto se crea estáticamente (en tiempo de compilación). Por ello se limita demasiado con respecto a los requisitos de manipulación de módulos programa, necesarios para la configuración dinámica de sistemas-programa. Para resolver este problema, se necesita el soporte de los lenguajes de programación para ver objetos como (6) en un modo tipo. Este soporte puede encontrarse en el concepto de *tipos de paquete* que ofrece una visión tipificada equivalente de todos los módulos-programa, desde el dato puro al código puro. En muchos lenguajes de programación, tales visiones se aplican solamente a datos, y los módulos de código suelen tratarse en un nivel inferior. En (7) se declara como tipo de paquete una cola de espera; esta declaración se debería comparar a la declaración de tipo de registro dada en (1). Objetos del tipo *complejo* como el (1),

```
tipo COLA-DE-ESPERA es                              (7)
  paquete
    procedimiento MECOLA-DE-ESPERA (X: ITEM);
    procedimiento SACOLA-DE-ESPERA (X: fuera
    ITEM);
  fin paquete;
```

tienen dos atributos reales (una visión orientada al dato), mientras que objetos del tipo *cola de espera* dado en (7) poseen dos atributos que son procedimientos (visión orientada a la operación). El concepto de tipo de paquete no tiene soporte en Ada, pero se adoptó una notación semejante para exponer mejor las ideas y ofrecer la potencia expresiva necesaria.

En cualquier tipo de objeto, un programador puede declarar objetos constantes, de valor fijo durante la vida del programa, y objetos variables, cuyos valores se alteran cuando el programa se ejecuta. Un objeto constante del tipo de registro *complejo* se puede declarar e iniciar así:

```
J: constante COMPLEJO: = (0,1),                    (8)
```

en cambio, un objeto constante del tipo de paquete *cola de espera* se instancia con la declaración dada en (9).

```
COLA-DE-ESPERA-COMO-ORDENACION:
COLA-DE-ESPERA constante =
  cuerpo paquete
    – Constantes privadas del paquete, tipos y datos
    LONG-MAX:        constante ENTERO = 100;
    VALOR:           ordenación (1..LONG-MAX)
                     del ITEM;
    LONG:            entero: = 0;
    CABEZA, COLA: entero: = 1;

    procedimiento MECOLA-DE-ESPERA
    (X: ITEM) es                                   (9)
    comienzo
      VALOR (COLA): = X;
      COLA: = (COLA mod LONG-MAX) + 1;
      LONG: = LONG + 1;
    fin MECOLA-DE-ESPERA;

    procedimiento SACOLA-DE-ESPERA
    (X: fuera ITEM) es
    comienzo
      X: = VALOR (CABEZA);
      CABEZA: = (CABEZA mod LONG-MAX) + 1;
      LONG: = LONG – 1;
    fin SACOLA-DE-ESPERA
  fin cuerpo paquete;
```

La sentencia de asignación (8) da un valor inicial, representado como agregado de registros, al objeto constante J del tipo *complejo;* la contenida en (9) asigna un valor inicial, representado como cuerpo de paquete, a un objeto constante COLA-DE-ESPERA-COMO-ORDENACION del tipo *cola de espera.* Los valores de ambos objetos permanecen invariables en toda su vida. Se introducen variables de ciertos tipos mediante declaraciones:

```
Z: COMPLEJO;
MI-COLA-DE-ESPERA: COLA-DE-ESPERA;                 (10)
```

En el tiempo apropiado, se puede poner o cambiar el valor de una variable, usando la bien conocida sentencia de asignación. Las variables de (10) pueden alcanzar el valor:

```
Z: = (RE ⩾ 5, IM ⩾ 7);                             (11)
```

```
MI-COLA-DE-ESPERA = COLA-DE-ESPERA-
COMO-ORDENACION;                                   (12)
```

En la asignación (11), su miembro de la derecha indica un agregado de registros que representa un valor del registro. La sentencia de asignación (12) es importante, ya que selecciona una posible realización del módulo-programa, accesible a través de la variable MI-COLA-DE-ESPERA. Si existiese una segunda realización (13) para objetos del tipo *cola de espera,* un gestor

```
COLA-DE-ESPERA-COMO-LISTA: COLA-DE-
ESPERA constante =
  Cuerpo paquete
  . . .                                            (13)
  . . .
  fin cuerpo paquete;
```

de configuración podría seleccionar, típicamente en tiempo de iniciación, la realización deseada.

Hasta aquí sólo hemos expuesto una forma de tratar uniformemente módulos-programa compuestos de dato y código, usando un concepto de tipo de dato abstracto, y una notación más o menos similar a la de Ada. Aunque los ejemplos son sencillos para facilitar la presentación de los conceptos, de hecho hemos descrito y realizado programas complejos, así como conjuntos bien escogidos de tipos de datos abstractos (p. ej., un editor orientado a pantalla y un programa de tratamiento de llamadas para un sistema de conmutación telefónica).

Módulo-programa sustituible

El concepto de tipo de dato abstracto es también muy útil para describir módulos-programa sustituibles. Además, se necesita el *atributos específicos del tipo* (las operaciones peculiares de la funcionalidad del tipo de dato abstracto) para introducir nuevas operaciones que permitan al gestor de reconfiguración transferir información de estado interno de la versión antigua a la nueva, como antes se indicó. Estas son los *atributos de reconfiguración.*

En (7) se dio un ejemplo de un tipo de paquete que sólo tiene atributos específicos del tipo. Para ser sustituible se debería mejorar el mismo tipo de paquete con un procedimiento delegado, que admita una posible renovación. Llamemos a este nuevo tipo de paquete COLA-DE-ESPERA-LLENA, como se muestra en (14). Siempre que haya que sustituir una primera realización por otra (segunda), el gestor de reconfiguración llama la entrada DELEGACION de la primera y pasa la segunda como parámetro. El resultado de una operación DELEGACION será que el estado interno de la primera realización se transfiera totalmente a la segunda. Una vez completada esta transferencia de estado, el gestor de

```
tipo COLA-DE-ESPERA-LLENA es
  paquete
    – tipo atributos específicos
    procedimiento MECOLA-DE-ESPERA (X: ITEM);
    procedimiento SACOLA-DE-ESPERA
    (X: fuera ITEM);                          (14)
    – atributos de reconfiguración
    procedimiento DELEGACION
    (CLL: en fuera COLA-DE-ESPERA-LLENA);
  fin paquete;
```

configuración conmuta a la nueva realización y destruye las referencias a la antigua.

En (15) se muestra una realización del tipo COLA-DE-ESPERA-LLENA:

```
COLA-DE-ESPERA-LLENA-COMO-ORDENACION:
COLA-DE-ESPERA-LLENA constante: =
  cuerpo paquete
    – Constantes privadas del paquete, tipos y datos
    LONG-MAX:      constante ENTERO: = 100;
    VALOR:         ordenación (1..LONG-MAX)
                   del ITEM;
    LONG:          entero: = 0;
    CABEZA, COLA:  entero: = 1;

    procedimiento MECOLA-DE-ESPERA (X: ITEM) es
    comienzo
      VALOR (COLA): = X;
      COLA: = (COLA mod LONG-MAX) + 1;
      LONG: = LONG + 1;
    fin MECOLA-DE-ESPERA;                     (15)

    procedimiento SACOLA-DE-ESPERA
    (X: fuera ITEM) es
    comienzo
      X: = VALOR (CABEZA);
      CABEZA: = (CABEZA mod LONG-MAX) + 1;
      LONG: = LONG – 1;
    fin SACOLA-DE-ESPERA;

    procedimiento DELEGACION (CLL: en fuera
    COLA-DE-ESPERA-LLENA) es
    comienzo
      mientras LONG: = 0 bucle
        CLL. MECOLA-DE-ESPERA
        (VALOR (CABEZA));
        CABEZA: = (CABEZA MOD LONG-MAX)
        + 1;
        LONG: = LONG – 1;
      fin bucle;
    fin DELEGACION;
  fin cuerpo paquete;
```

Consideremos ahora el módulo de reconfiguración. Supongamos que una variable COLA-DE-ESPERA-ACTUAL referencia la realización actual de la cola de espera, que todavía podría ser el paquete COLA-DE-ESPERA-COMO-ORDENACION-LLENA dado en (15). Se supone también que existe otra realización COLA-DE-ESPERA-COMO-LISTA-LLENA. Usando una variable COLA-DE-ESPERA-LLENA-TEMP, se puede escribir un extracto de módulo de reconfiguración (16), que puede explicarse

```
declarar
  COLA-DE-ESPERA-LLENA-TEMP:
  COLA-DE-ESPERA-LLENA;
comienzo                                      (16)
  COLA-DE-ESPERA-LLENA-TEMP: = COLA-
  DE-ESPERA-COMO-LISTA-LLENA;
  COLA-DE-ESPERA-ACTUAL. DELEGACION
  (COLA-DE-ESPERA-LLENA-TEMP);
  COLA-DE-ESPERA-ACTUAL: = COLA-DE-
  ESPERA-LLENA-TEMP;
fin;
```

así con referencia a la figura 7: la primera sentencia de asignación da cuerpo a una nueva realización (Fig. 7b); llamando al procedimiento DELEGACION se transfiere la información de estado interno del antiguo al nuevo módulo (Fig. 7c), mientras que la segunda sentencia conmuta de la primera realización a la segunda (Fig. 7d). En este punto, la sustitución es completa.

Verificación experimental

El concepto teórico de la modificación dinámica de un programa, usando la noción de tipos de datos abstractos con atributos específicos del tipo y de reconfiguración, se ha comprobado en un sistema de microprocesador de tecnología actual, el Intel iAPX432[3]. Este experimento indicó la factibilidad del modelo seleccionado y demostró el interés de diseñar sistemas-programa de forma modular por medio del concepto de tipo de dato abstracto.

Referencias

1 Intel Corporation: iAPX432 General Data Processor Architecture Reference Manual: Santa Clara, Intel, 1981.

2 R. T. Boute, J. De Man y H. Peeters: Secure On-The-Fly Software Modification: *Proceedings of the Fourth International Conference on Software for Telecommunication Switching Systems,* Coventry, 20–24 julio 1981, Institution of Electrical Engineers Conference Publication n° 198, 1981, págs. 49–53.

3 J. De Man: Experiments with Dynamic Software Modification: *Proceedings of EUROMICRO 82, 8th Annual Symposium on Microprocessing and Microprogramming,* Haifa, 6–8 septiembre 1982.

J. A. De Man se graduó en electrónica en 1976, por la Universidad de Gante. Ha realizado un compilador para MISTRAL, un lenguaje orientado a la máquina. En 1979, trabajó en el proyecto iAPX432 en Intel Corp., Aloha, Oregon. Desde 1981 dirige un grupo de diseño en el departamento de programación y sistemas avanzados de BTM; ese grupo ha preparado un compilador CHILL mediante modernas herramientas soporte y especificaciones formales.

Marcel A. E. van Rumste obtuvo un grado en ingeniería eléctrica en la Universidad de Gante, en 1975. Dos años después se unió a un equipo en el laboratorio central de BTM, trabajando en diseño de placas de circuito impreso y VLSI con ayuda de ordenador. Actualmente el Sr. van Rumste dirige el departamento de programación y sistemas avanzados, dedicado a la especificación, diseño y realización de sistemas de ordenador en tiempo real con tolerancia a fallos.

En este número

Kidd, E. R.
La programación en ITT
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 276–283

Cada vez hay más productos y servicios importantes de ITT que dependen de la programación avanzada, figurando entre ellos los de telecomunicación y electrónica, seguros y finanzas y productos manufacturados. Comenzando en 1956 con un ordenador experimental, la actividad de la Compañía en programación ha crecido hasta abarcar una red mundial, dirigida por centros especializados en Europa y Norteamérica. Este artículo expone técnicas de gestión desarrolladas por ITT, incluyendo directrices para la productividad y calidad de la programación, y un grupo de "Early Warning" (información anticipada). Se analizan también las tendencias futuras en cuanto a reutilización, prevención y eliminación de defectos, facilidad de uso y suministro de programas y equipos por separado.

Lawson, D. A.
Desarrollo planificado de la programación ITT 1240 por diferentes centros
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 284–288

La coincidencia en la producción de programas para la central digital ITT 1240 en diferentes lugares del mundo, crea una serie de problemas de gestión. Este artículo describe un camino para solucionar estos problemas, así como el proceso analítico empleado en la partición del sistema para un eficaz desarrollo concurrente. Se hace hincapié en la importancia de planificar los interfaces antes del desarrollo, y luego en el control de cambios durante el mismo con objeto de limitar las interacciones. Se muestra cómo el control distribuido de la central ITT 1240 hace realizable este método. Se explica la necesidad de agrupar los cambios en ediciones cuidadosamente planeadas para limitar las pruebas de regresión y dar estabilidad al sistema.

Saward, M. P.
Herramientas para soporte de la programación
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 289–294

Los sistemas soporte y las herramientas de programación son fundamentales para el desarrollo de cualquier producto con alto contenido de programación. El autor describe algunas de las necesidades y beneficios de una organización que prepara un desarrollo semejante. Trata los procesos de desarrollo, producción, fabricación e integración de la programación. También se describe cómo se han de hacer las inversiones y qué factores deben ser considerados, y en particular cómo aborda el Programming Support Centre la tarea de distribuir las herramientas del sistema ITT 1240 a los centros ITT.

Jones, T. C.
Prevención y eliminación de defectos en programación
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 295–300

Prevenir y eliminar defectos en los programas utilizados en grandes sistemas gobernados por ordenador, como son los de conmutación telefónica, es costoso y consume tiempo. La ineficacia de los métodos tradicionales de prueba de programas para detectar defectos, ha llevado a desarrollar nuevas técnicas. El autor describe estos nuevos métodos, que incluyen diseño formal e inspecciones de código, el uso de pruebas de exactitud, mejoras en los lenguajes de programación y construcción de prototipos o modelos. Con las técnicas más recientes, la eficacia en la eliminación de defectos puede superar el 95%, acercándose con ello a una programación cero-defectos.

Dunn, R. H.; Ullman, R. S.
Garantía de calidad para programas de ordenador
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 301–306

Los programas de ordenador aumentan su complejidad, a medida que las funciones que tienen que realizar son más sofisticadas. Así, a la vez que crece la probabilidad de introducir defectos en los programas en la fase de desarrollo, más importantes son las consecuencias de tales defectos. Los autores examinan la forma de reducir estos riesgos al mínimo, siguiendo procedimientos estrictos de garantía de calidad durante el citado desarrollo. Ello implica el establecimiento de grupos de calidad de programación, que trabajen muy unidos a los grupos de desarrollo, pero sin depender de ellos. Tales grupos pueden ayudar a la dirección a controlar los proyectos, representar los intereses de los usuarios y mejorar la calidad de los programas.

Holt, A. W.; Ramsey, H. R.; Grimes, J. D.
La tecnología de sistemas de coordinación como base para un entorno de programación
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 307–314

Este artículo presenta los conceptos básicos que implica un proyecto de entorno de programación de ITT. Como en cualquier proyecto de este tipo, se necesita el desarrollo de herramientas y de medios especializados para dar apoyo a las tareas relacionadas con la programación. Sin embargo, el proyecto acomete también problemas mucho más importantes, en cuanto a la coordinación de grupos organizados de personal en actividades que precisan de cooperación mutua. El artículo describe dos aspectos originales del tratamiento de este problema por ITT: la teoría rol/actividad y los sistemas de coordinación. Se presentan estas dos ideas, analizando su aplicación a los entornos de trabajo, en general, y a los entornos de programación, en particular.

Moody, H. F.; Wilson, F.
Entorno de aprendizaje interactivo para la enseñanza de la programación
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 315–319

Al ir haciéndose más complejos los sistemas de comunicación, necesitando siempre un volumen de programación mayor, se agranda el problema de educar a los programadores y mantenerlos en línea con los avances tecnológicos. Para ello se está desarrollando un nuevo sistema interactivo, con videodisco, que proporcionará cursos individualizados, capaces de adecuarse a la velocidad de aprendizaje del estudiante. El artículo describe las características principales de este sistema interactivo, en esencia constituido por dos reproductores de videodisco controlados por un microordenador.

Kimbleton, S. R.; Wang, P. S.-C.
Soporte de comunicación ITT/NET para el desarrollo de programas
Comunicaciones Eléctricas (1983), volumen 57, nº 4, págs. 320–325

Una compañía multinacional como ITT, donde el desarrollo de los proyectos más importantes puede repartirse entre unidades de distintos países, precisa un eficaz soporte de comunicación. Los autores describen el prototipo ITT/NET diseñado con este objetivo. Ventajas importantes de esta red son su adaptabilidad a cambios en los requisitos de los usuarios y la independencia que da a las unidades de ITT respecto a adquisición de equipos y parametrización y generación del sistema. Proporciona soporte de operaciones, de usuario y de bloques constructivos a los distintos centros. En los próximos años, se desarrollará un conjunto completo de herramientas para la programación multicentro, a fin de crear un entorno de desarrollo integrado para todos los proyectos de programación de ITT.

Haque, T.
Entorno de programación CHILL para el ITT 1240
Comunicaciones Eléctricas (1983), volumen 57, n° 4, págs. 326–333

ITT se comprometió, desde el principio, a usar el CHILL, lo que ha influido fuertemente en el entorno de programación del ITT 1240. La central digital ITT 1240 se estructura en un gran número de elementos de control distribuidos, en los cuales se cargan programas de aplicación o máquinas soporte. El diseño de alto nivel, apoyado por la biblioteca que describe los componentes de programación junto con la metabase de datos distribuida del ITT 1240, asegura la sólida construcción de estas máquinas soporte. La gestión de la base de datos se lleva a cabo a través de un lenguaje de manipulación de datos inmerso en el lenguaje CHILL. Se considera también la traducción de la fuente CHILL, su compilación y prueba. Finalmente, el artículo examina los diferentes sistemas soporte para ingeniería de aplicación y producción de programas, concluyendo con un informe sobre el estado actual del CHILL.

Raible, R.
Biblioteca de proyecto
Comunicaciones Eléctricas (1983), volumen 57, n° 4, págs. 334–338

La documentación completa es un aspecto esencial del desarrollo de programas complejos. El autor describe una biblioteca de proyecto, utilizada en ITT como soporte de la organización y administración de toda la documentación relacionada con los productos y subproductos de un proyecto de desarrollo de programación. Además, la biblioteca de proyecto facilita el control y planificación del mismo, proporcionando secuencias normalizadas y procedimientos de entrega para los subproductos. Los resultados obtenidos hasta ahora han estimulado el uso más amplio del concepto de biblioteca de proyecto para futuros desarrollos.

De Man, J. A.; Van Rumste, M. A. E.
Modificación dinámica de programas
Comunicaciones Eléctricas (1983), volumen 57, n° 4, págs. 339–346

Se han dedicado considerables esfuerzos de diseño al desarrollo de equipos que puedan ser sustituidos sin efectos apreciables sobre la operación global del sistema. No se considera tanto, en cambio, la importancia que tiene el diseñar programas que puedan sustituirse dinámicamente. El autor examina los principales factores necesarios para conseguir esta meta. Son importantes el concepto de módulo, al igual que en el equipo, y el concepto de tipo de dato abstracto. Los conceptos de modificación dinámica de programas tratados en este artículo, se han comprobado en un sistema microprocesador de tecnología actual.

Oficinas Editoriales

La correspondencia relacionada con las diferentes versiones de Electrical Communication debe dirigirse al editor correspondiente:

Michael Deason
Electrical Communication
Great Eastern House
Edinburgh Way
Harlow, Essex
England

Otto Grewe
Elektrisches Nachrichtenwesen
Hellmuth-Hirth-Strasse 42
7000 Stuttgart 40
Deutsche Bundesrepublik

Antonio Soto
Comunicaciones Eléctricas
Ramírez de Prado, 5
Madrid – 7
España

Lester A. Gimpelson
Revue des Télécommunications
ITT Europe Inc.
Avenue Louise 480
1050 Bruxelles
Belgique