Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

DESENVOLVIMENTO E TESTES DE PROGRAMAS ESTRUTURADOS

Eduardo Ferreira Eleotério
CELEPAR
Rua Mateus Leme, 1561 - Curitiba - PR - Brasil

INTRODUÇÃO

Em uma Empresa de Processamento de Dados, normalmente encontramos os mais variados métodos de se confeccionar um programa, no qual irá impactar diretamente em futuras manutenções dos mesmos.

Em estudos realizados ao longo do tempo nos mostrou que esta variação era obtida através de métodos próprios adquiridos pelos programadores, dificultando a alteração de um programa que tivesse sido feito por um outro programador.

Preocupados em unificar o desenvolvimento de programas, pesquisamos e chegamos a conclusão que seria necessário a aplicação de um método que utilizado por vários programadores, obtivessemos uma solução única. Dentro dos estudados, concluímos que seria adequado para nossas necessidades, os métodos de programação estruturados, no qual obtivemos pleno êxito em nossas aplicações. E este trabalho mostra como desenvolvemos e testamos programas de forma estruturada.

I - ESTUDO DA DEFINIÇÃO

Esta etapa dos procedimentos do programador tem como objetivo, propiciar ao mesmo tempo, um conhecimento detalhado e seguro dos dados a serem manuseados pelo programa, e dos procedimentos necessários para a transformação destes.

Para tanto, é necessário que a definição esteja completa, pois, é importante que se tenha todas as informações para o desenvolvimento do programa.

Ao estudar a definição, o programador deverá entender a lógica do programa, anotando à parte todas as dúvidas que surgirem, fazer um levantamento e esclarecer todos os pontos obscuros. Estes procedimentos devem ser feitos, tantas vezes quanto for necessário, até que tenha compreendido a de

finição por completo, evitando assim interpretações errôneas.

Como o programa será desenvolvido através de um método estruturado, o programador deverá ter uma preocupação adicional com os dados de entrada e saída do seu programa, sendo este procedimentos fundamental para a aplicação do método.

Como resultado final desta etapa, o programador terá o controle e conhecimento necessário do programa, facilitando sobremaneira a continuidade do desenvolvimento.

II - ESTRUTURAÇÃO LÓGICA

A finalidade da estruturação lógica, é obter solução lógica para o problema representado pelos dados a serem processados, e os procedimentos necessários para esta transformação. E, para obter esta solução, devemos utilizar um método estruturado, no qual nos solucione o problema sem que haja a preocupação a nível de detalhe.

Dentro de experiências realizadas em nossa empresa, chegamos a conclusão que o método de JACKSON supria bem nossas necessidades, pois consiste em estruturar um programa baseado nos dados de entrada e saída do programa, e mostrou também uma simbologia variada e de fácil manuseio.

Outra vantagem do método, é a uniformização e padronização que se vai adquirir nos programas, facilitando sobremaneira as futuras manutenções destes.

III - PLANEJAMENTO DOS TESTES DO PROGRAMA

O planejamento deve ser feito de maneira criteriosa e objetiva. Devem ser aproveitados os recursos disponíveis como ferramenta de apoio ao programador, sempre observando o uso otimizado do computador.

Um plano de teste bem elaborado, servirá de guia para o programador, quando da confecção dos arquivos e de todo complexo de testes que o programa irá precisar.

No entanto, este planejamento poderá não ser o definitivo, pois a medida que outros passos forem desenvolvidos, a assimilação aumentará, podendo surgir novas situações, que deverão ser incluídas no mesmo. Ao final da codificação este deverá ser obrigatoriamente revisto.

O plano de teste deverá incluir atividades como:

01. REVISÃO DO PROGRAMA

O primeiro procedimentos para assegurar a correção do programa é a revisão estática da codificação. Deverão portanto ser planejadas as revisões a serem efetuadas.

Existem vários processos de revisão, sendo os mais comuns, a revisão do código, o teste de lógica e a revisão formal, que é também conhecida como WALK-THROUGH.

A revisão do código e o teste de lógica, devem ser aplicados para todos os programas, e a revisão formal, apenas para os programas que a justifiquem, por tratar-se de um processo caro.

02. ETAPAS DE TESTES A SEREM EXECUTADAS

De acordo com os procedimentos dos programas, devemos planejar as etapas de testes necessários. Esta informação nos dirá quantos arquivos serão necessários para o teste do programa.

Para exemplificar, vamos tomar como base um programa atualizador de cadastro, considerando a existência de um arquivo movimento com apenas 1 registro de mesma identificação no cadastro. Teríamos as seguintes etapas:

Etapa 1 - Movimento com exclusões e alterações para arquivo cadastro vazio.

Etapa 2 - Movimento com inclusões para cadastro vazio.

Etapa 3 - Movimento com inclusões, alterações e exclusões corretas para cadastro criado na etapa 2.

Etapa 4 - Movimento com condições inválidas; inclusão para chave existente no cadastro criado na etapa 3.

Estas etapas mostram de uma maneira geral como podemos planejar as condições de testes, sendo que para cada procedimento teremos condições diferentes, e servirá como apoio ao planejamento dos arquivos a serem utilizados.

03. PLANEJAMENTO DOS ARQUIVOS DE ENTRADA

Planejar o conteúdo dos arquivos que foram identificados como necessários para o teste completo verificando o fator qualitativo e quantitativo dos mesmos.

Os arquivos podem ser planejados da seguinte maneira:

03.01. ARQUIVOS CORRETOS

Deve ser planejado para que satisfaça todas as condições de passagem correta de um programa. Os arquivos devem ser projetados, de acordo com sua definição devendo, em termos de conteúdo, se aproximar o máximo possível de um arquivo real que será utilizado em produção. Para tanto algumas condições devem ser lembradas:

- GERAL
  - . Arquivos com apenas um registro;
  - . Arquivos sem registro;
  - . Limites máximos e mínimos de conjuntos de registros;
  - . Registros de arquivos variáveis, com tamanho máximo, mínimo e intermediário.

- MERGE DE ARQUIVOS
  - . As identificações tenham condições de igualdade e desigualdade;
  - . Um arquivo termina primeiro e vice-versa;

. Todas as identificações de um arquivo iguais a dos outros bem como todos desiguais.

- CONSISTÊNCIA E COMPATIBILIDADE
  . Todos os campos deverão estar corretos;
  . As consistências cruzadas devem estar corretas;
  . Todos os fechamentos devem estar corretos.

- CÁLCULO
  . Todos os valores numéricos, prevendo valores mínimo, máximo e intermediários.

- ATUALIZAÇÃO
  . Código de atualização correto, contendo inclusões, exclusões e alterações sempre corretas.

- FORMATAÇAO
  . Todos os tipos de registros de erão estar corretos.

- RELATÓRIOS
  . Quantidade de registros que preencham mais de uma página;
  . Quantidade de registros que não preencham uma página;
  . Quantidade de registros que terminam exatamente no limite da página.

- TABELAS
  . Registros que testem os limites dos indexadores e subscritos do programa;
  . Registros que testem as tabelas internas do programa.

03.02. ARQUIVOS ERRADOS

Deve ser planejado para que force o programa a identificar o erro. Os arquivos devem ser projetados de maneira que o programa seja testado quanto a condições anormais e absurdas, que raramente ocorrem e que não foram previstas pelo sistema, causando o cancelamento de um programa. Ainda, estes arquivos devem conter erros previstos pelo programa, forçando a verificação dos testes do mesmo.

- GERAL
  . Ausência e duplicidade de HEADER's, TRAILLER'S, CAPA DE LOTE, REGISTRO DE FECHAMENTO e outros do mesmo nível;
  . Arquivos com apenas 1 registro;
  . Arquivos sem registros;
  . Erros de parâmetros informados.

- MERGE DE ARQUIVOS
  . Prever as condições de erros de combinação de identificações entre os vários arquivos;

. Identificação, fora de sequência.

- CONSISTÊNCIA E COMPATIBILIDADE
  . Deve ter todos os campos consistidos com o mais variado tipo de erro;
  . As consistências cruzadas devem estar incorretas;
  . Todos os fechamentos devem estar incorretos.

- ATUALIZAÇÃO
  . Código de atualização incorreto;
  . Exclusão e alteração para não existentes;
  . Inclusão para já existente.

- FORMATAÇÃO
  . Registros fora de sequência ou agrupados incorretamente.

03.03. ARQUIVOS MISTOS

Apesar do programa já ter sido planejado para executar estas funções, um arquivo que podemos chamar de misto, pode propiciar mais uma condição diferente de teste. Dependendo da situação, pode-se utilizar os arquivos anteriormente planejados, pois as condições de testes serão iguais aos descritos no item 03.01 e 03.02.

04. PLANEJAMENTO DOS RESULTADOS

Para cada etapa deverão ser esboçados todos os resultados esperados em função dos arquivos de entrada. Isto servirá para facilitar ao programador a conferência dos arquivos e relatórios de saída gerados pelo programa. Normalmente utiliza-se o processo de conferência visual, que é cansativo, o que prejudica ao longo dos testes a conferência de todas as condições. Planejando-se os resultados poderemos utilizar conferência automática o que, além de facilitar, dará ainda um grau de confiabilidade maior nos resultados finais.

05. DOCUMENTAÇÃO DO PLANO DE TESTE

Tão importante quanto fazer o plano de testes é que estes sejam documentados e arquivados de uma maneira que permitam sua utilização quando de uma manutenção futura no programa, sem necessidades de refazê-lo.

IV - CODIFICAÇÃO

Consiste em transformar em códigos, que possam ser interpretados pelo computador, a solução lógica do programa.

A codificação do programa deverá ser feita de forma parcial, que tem a particularidade de codificação e testes simultâneos, ou seja, codifica-se a rotina principal e testa seu funcionamento, quando estiver correto, passará aos testes dos vários grupos de rotinas individualmente, e só se

executando o próximo, quando o anterior estiver correto.

Esta prática, associada a uma necessária padronização às normas de codificação, tem a vantagem do programa se tornar menos cansativo, pois o programador variará constantemente seu serviço, além de que, as rotinas principais estarão exaustivamente testados, aumentando a confiabilidade do programa.

É importante que o programa seja codificado de maneira clara e explicativa, facilitando futuras manutenções.

V - REVISÃO

A revisão sempre é necessária quando se pensa em otimizar ou melhorar alguma coisa. Neste ponto devemos fazer a revisão de tudo que já temos de concreto, e no caso, o programa e o planejamento do teste. A revisão sempre amplia a visão do programador em torno do que está sendo feito, dando-lhe mais segurança e confiabilidade.

Apesar do programador ter feito o fluxo lógico do programa, e codificado de acordo com ele, pode ter passado alguma condição desapercebida, ou a maneira como seu programa foi escrito, pode não ter sido a melhor.Ainda há a possibilidade da existência de erros na transcrição que poderão influenciar a lógica, sem ter aparecido erro de sintaxe na compilação. A revisão além de ser um ótimo apoio para descobrir estes erros, economizará um bom número de testes que seriam necessários processar no computador. Propomos os seguintes procedimentos para revisar um programa:

01. REVISÃO DO CÓDIGO

É um método bastante informal que deve no entanto obedecer uma lista de procedimentos para se assegurar que não foi esquecido nada. É executado pelo programador com ou sem ajuda de um terceiro, a critério deste.

O programador de posse de uma listagem simples do programa, tentará encontrar inicialmente erros de codificação e transcrição confrontando o programa com os diagramas de estrutura lógica e as folhas de codificação, a fim de assegurar que o programa foi codificado de acordo com a lógica desenvolvida e transcrito de acordo com o que foi codificado.

02. TESTE DE LÓGICA

Consiste na prática de seguir a lógica do programa através do código, simulando registros que atendam as condições do programa. O uso de um terceiro é aconselhável, pois este poderá auxiliar bastante no acompanhamento de processamento dos registros no programa. Sendo isto na verdade um teste normal, devem ser testadas todas as condições possíveis, sempre em obediência ao plano de teste elaborado.

03. REVISÃO FORMAL

A revisão formal se caracteriza pela existência de um método e pelo desenvolvimento formal de outras pessoas no processo, ficando o revisado

praticamente em situação de expectador. Possui como grande vantagem sobre as revisões informais o fato de além de provar a correção dos programas, servir como instrumento de divulgação de conhecimentos entre o pessoal, bem como revisar constantemente a metodologia de desenvolvimento de programas através das idéias que surgem no trabalho de grupo.

Por ser um processo caro, nem sempre compensa uma revisão deste tipo, devendo ser observados critérios para determinar sua necessidade.

03.01. DETERMINAÇÃO DA NECESSIDADE

A revisão formal deve ser efetuada sempre que o programa possua procedimentos mais complexos ou numerosos, para os quais normalmente não conseguiríamos obter todas as condições de testes necessários, não devendo ser aplicada a programas pequenos ou fáceis, a menos que existam sérias limitações com relação ao uso do computador para teste.

A solicitação da revisão pode partir do programador que desenvolveu o programa, sua chefia ou de elementos diretamente ligados ao sistema em desenvolvimento.

03.02. PARTICIPANTES E SUAS RESPONSABILIDADES

Todos os participantes têm no processo, responsabilidades iguais perante a correção do programa, uma vez que as decisões devem ser tomadas em consenso pelo grupo. Todos devem estar familiarizados com o método e dispostos a ajudar.

Na escolha dos revisores deve ser incluído pelo menos, sempre que possível, um elemento que participa do mesmo projeto podendo ser programador ou analista, um programador experiente e um de menor experiência que não participem necessariamente do projeto.

Por uma questão de organização, definem-se três tipos de participantes:

- O Coordenador da Revisão, que é o responsável por coordenar o grupo, e que tem basicamente como atribuições:
  - Determinar e avisar os participantes da data, hora e local para a revisão;
  - Encaminhar o material necessário aos participantes;
  - Coordenar a sessão, mantendo a ordem, orientando para que cada um fale de uma vez, manter a discussão rigorosamente sobre o assunto;
  - Opinar e orientar o grupo para suas decisões.

- Os revisores, que devem participar do processo, obedecendo as determinações do Coordenador de Revisão, com o objetivo único de opinar acerca da matéria em questão. Deve haver um número entre 1 e 3 revisores, dependendo da complexidade do programa.

- O Revisado, que deve simplesmente assistir ao processo, sem direito a opinar, tendo como única atribuição corrigir os erros apontados pelos revisores.

03.03. PROCEDIMENTOS

O Coordenador da Revisão deve encaminhar com antecedência o material necessário aos participantes, que deve incluir a última compilação sem erros , diagrama de estrutura, lay-outs, etc, material este que cada participante deverá revisar sozinho antes da reunião, anotando os pontos que julgue errado ou obscuro.

A reunião deve ser conduzida de modo a que os participantes apontem os problemas, porém sem resolvê-los ou sugerir soluções, anotando-se cada observação aprovada em consenso pelo grupo. Deve ser realizada em local isolado onde não haja interrupções de espécie alguma, sendo desejável a existência de quadro, flip-chart ou outros meios de apoio. Sua demora não deverá exceder a duas horas.

Cada participante deve fazer ao final da revisão ao menos um comentário positivo e um negativo, sempre ao programa e nunca ao programador. Todas as partes do programa devem ser revisadas e na sua conclusão o Coordenador da Revisão deve preparar um relatório com o sumário das observações e entregar ao programador para o acerto do programa. Este relatório final da revisão deve ser claro, de maneira que qualquer pessoa possa entendê-lo sem ter participado da revisão, bem como deve ser arquivado com a documentação do programa.

A revisão pode acarretar em aprovação do programa ou solicitação de nova revisão. No último caso, após os acertos do programa, este deve ser novamente revisado, devendo-se porém, evitar repetição dos comentários feitos anteriormente.

04. LISTA DE VERIFICAÇÃO

Para se estabelecer uma linha de ação sobre os pontos a verificar dentro de um programa, é aconselhável que em cada empresa, de acordo com sua realidade e com as linguagens mais utilizadas, desenvolver seu próprio método. Como modelo, podemos citar para linguagem COBOL:

04.01. IDENTIFICAÇÃO DIVISION

. O nome do programa e quadro de explicações;
. Validade da função descrita para o programa.

04.02. ENVIRONMENT DIVISION

. Os nomes externos dos arquivos correspondem ao pedido;
. Ausência ou duplicidade de algum arquivo.

04.03. DATA DIVISION/FILE SECTION

. Existência de FD para arquivos comuns e SD para arquivos SORT;
. Correspondência com os nomes da SELECT;
. Presença da cláusula "RECORD CONTAINS" para forçar a verificação do tamanho do registro definido;
. As definições dos arquivos correspondem a sua característica física;

. Definição dos registros.

04.04. DATA DIVISION/WORKING-STORAGE SECTION

. Os acumuladores numéricos e inicializados;
. Caracter de controle válido nas linhas de impressão;
. Definição dos registros;
. Números de nível, para evitar que um item grupo abranja maior ou menor número de campos que o previsto;
. Tamanho de subscritos, comportam as ocorrências máximas;
. Literais estão escritos corretamente.

04.05. PROCEDURE DIVISION

. Parâmetros passados para as subrotinas chamadas pelo programa estão no formato e ordem requeridos;
. Existência de LINKAGE SECTION e PROCEDURE DIVISION USING..., quando o programa recebe parâmetros;
. Existe um comando SET inicializando o indexador referente a tabela que se vai ser pesquisada, quando do uso de SEARCH sem opção ALL;
. Se não está sendo usado o indexador de uma tabela em outra;
. Validade dos campos usados em cálculos, inclusive validade de suas fórmulas;
. Todas as sections tem a palavra SECTION e terminam com EXIT;
. Todos os arquivos são abertos/fechados corretamente;
. Alinhamento de IF's procurando ausência ou excesso de pontos;
. A área receptora está correta na utilização do comando STRING;
. Tamanho de áreas do READ INTO e WRITE FROM, estão de acordo com o definido na FILE SECTION;
. DATA/HORA, quando necessários são obtidos como primeiras instruções;
. Parâmetros são validados quanto a erro ou ausência;
. O uso correto da palavra NOT em condições compostas;
. Em ninhos de comparação para cada IF existe o ELSE correspondente;
. Inexistência da possibilidade de leitura após fim de arquivo , inicialização e reinicialização corretos de contadores, acumuladores, tabelas e chaves;
. A última quebra do programa é efetuada na condição de término;
. O primeiro e último registro de qualquer arquivo é processado corretamente;
. GO TO para fora de sections;
. Referência a áreas de saída logo após um WRITE ou a áreas de entrada antes de um READ;
. Os literais estão escritos corretamente;
. Existem comandos que não serão executados;
. Existem DATA-NAMES que não estão sendo usados;

. Se o programa possui SORT interno:
OPEN-CLOSE e STOP RUN fora das sections da INPUT/OUTPUT PROCEDURE.
Os campos de classificação estão corretos (tamanho, posição,etc)
Existe desvio sem retorno para fora da INPUT/OUTPUT procedures.

VI - PREPARAÇÃO DOS ARQUIVOS

Na preparação física dos arquivos, deverão ser identificados quais os recursos disponíveis para sua criação bem como os meios de armazenamentos possíveis de utilizar. Dentre estes, o programador deve selecionar quais são os que melhor atendem as suas necessidades em função das características dos arquivos. Citamos alguns meios que podem ser utilizados:

01. CODIFICAÇÃO DE REGISTROS

Este recurso deve ser utilizado apenas quando o volume de dados é pequeno, pois ao contrário torna-se uma tarefa bastante estafante. Quando possível, deve ser feito diretamente utilizando-se documentos de entrada , que auxiliarão bastante, já que estes possuem formatos apropriados. Após sua codificação deverão ser transcritos para cartões ou meios magnéticos apropriados.

02. RECURSOS DE HARDWARE

Alguns computadores aceitam entrada e/ou alteração de dados diretamente por equipamento periférico. No caso de alimentação de dados deve ser aproveitado apenas para arquivos pequenos, pois é semelhante ao caso anterior. Já a alteração de dados de arquivos existentes, é uma prática aconselhável, pois o programador terá controle da alteração no momento que está processando. As limitações deste recurso devem ser avaliadas antes de optar pelo seu uso.

03. RECURSOS DE SOFTWARES

No mercado existe um bom número de softwares geradores de arquivos de testes. Eles tem demonstrado um bom grau de eficiência e apoio à programação, e podem ser processados juntos com o programa ou criando arquivos isoladamente. A pouca utilização destes softwares ocorre por falta de conhecimento do uso e potencialidade de condições que pode oferecer. Seu uso deve ser incentivado ao máximo, pois na maioria deles, a geração de um arquivo é rápida.

Praticamente todos os equipamentos já trazem um conjunto de utilitários que facilitam a tarefa de criação de arquivos.

04. PROGRAMAS ESPECIAIS

Sua utilização ocorre quando o arquivo é mais complexo e não pode ser gerado manualmente ou algum software disponível. Neste caso, codifica -

se um programa, normalmente elementar, que fará a criação de registros para o arquivo no formato desejado. Este programa, por ser temporário, não necessita seguir qualquer padronização e pode, por conseguinte ser escrito em pouco tempo.

05. O PRÓPRIO PROGRAMA

Este é um recurso que pode ser utilizado trazendo bons resultados. Após a leitura dos arquivos são transformados os registros lidos, formatando-os para as condições desejadas.

Na inexistência de um arquivo para formatar, é possível substituir os comandos de leitura por procedimentos que criem os registros com as condições desejadas.

06. COMBINAÇÃO DE RECURSOS

Muitas vezes pode ocorrer que um dos recursos existentes não consiga criar um arquivo conforme desejado. Neste caso podemos combinar o que dispomos, no sentido de que um determinado recurso crie um arquivo primitivo e outros o transformem para as condições necessárias a sua utilização no programa.

VII - EXECUÇÃO DOS TESTES

Consiste em processar o programa para cada etapa planejada, no sentido de apontar os erros de lógica. Se houver erro, o programa deve ser corrigido e executado novamente para esta condição. Só depois de se obter os resultados esperados é que a etapa seguinte deverá ser processada, até que a última seja executada.

01. TESTE PROGRESSIVO

O teste progressivo é o método que se aplica quando a codificação do programa é feita de forma parcial. Aqui, o teste se processa simultaneamente com a codificação do programa; primeiro codifica-se a estrutura principal de controle das rotinas, sem se preocupar com as rotinas de procedimentos específicos e testa-se a correção da lógica. Depois, os conjuntos de rotinas de procedimentos serão agregados ao programa nas etapas previstas no plano de teste. O importante é que somente se agregue ao programa um novo conjunto quando o que estiver em teste for considerado correto.

O que se obtém de vantagem com isto é que os erros poderão ser rapidamente localizados porque devem estar contidos apenas nas rotinas adicionadas, pois já sabe-se que as anteriores estão corretas.

Quando um programa é extenso e o programador codifica-o completamente, é normal que tanto sua produtividade quanto a previsão deminuam com o tempo por ser uma tarefa consativa. Por outro lado, alternando-se codificação e testes irão terminar estes problemas além do que haverá uma motivação maior por começar a obter resultados mais rápidos.

VIII - VERIFICAÇÃO DOS RESULTADOS

A escolha dos meios para verificação dos resultados, deve ser feita no sentido de facilitar ao programador a conferência das saídas geradas pelo programa. Como geralmente ocorre, estas conferências são feitas visualmente, sendo este processo bastante cansativo, nem sempre motivando o programador a conferir todas as condições testadas. Com ajuda de alguns recursos, estas conferências poderão ser efetuadas com um esforço humano menor, apresentando um grau de confiabilidade bem maior.

01. PRÉ-FORMATAÇÃO DE ARQUIVOS

A pré-formatação consiste em criar em um arquivo ou no papel, a imagem real do arquivo ou relatório de saída. Obtida como resultado do planejamento dos testes, este é o primeiro passo para a utilização de outros recursos, que permitirão uma melhor conferência visual ou automática, além de mostrar antecipadamente a imagem do resultado final do programa.

02. CONFERÊNCIA VISUAL

É a prática mais utilizada, sendo necessário apenas listar os arquivos e relatórios gerados pelo programa e conferir os resultados em função dos arquivos de entrada. Para cada execução de teste, todos os registros devem ser novamente conferidos, o que demanda um longo tempo. Para facilitar a conferência, o programador poderá fazer um programa listador dos arquivos, que vai separar seus campos, facilitando a identificação dos mesmos, ou até mesmo utilizar algum software com a mesma função. Se existe a pré-formatação dos arquivos, esta tarefa é bem mais rápida pois o único trabalho do programador é comparar o resultado do programa com o pré-formatado, sem se preocupar com o que o programa deveria processar.

03. CONFERÊNCIA AUTOMÁTICA

É necessário ter sido feita pré-formatação dos arquivos de saída e que esta tenha sido gravada em algum meio magnético. Com a utilização de programas especiais ou softwares de verificação quando disponíveis, compatibiliza-se os dois arquivos, o gerado pelo programa e o pré-formatado, emitindo um relatório que aponte todas as diferenças; se não houver, significa que o arquivo gerado está certo.

A melhor utilização deste método ocorre na proporção em que aumenta o volume dos dados gerados pelo programa em teste. Na prática é um recurso ainda pouco utilizado, porém pela economia que proporciona ao otimizar o tempo gasto pelo programador e pela confiabilidade que apresenta, certamente é um método que tende a se difundir.

04. FACILIDADES DAS LINGUAGENS

Nas linguagens de programação de alto nível, existem funções de apoio ao programador. Estas funções podem ser de grande auxílio, pois mostra em geral o andamento de um programa quando em execução. Temos comandos de DEBUG, bastante utilizados quando ocorre um erro mais difícil de identificar,

que consistem em mostrar os caminhos passados pelo programa. Outros que mostram quantas vezes cada comando foi executado, apresentam estatística que ajudam a otimizar o programa, mostrando onde ele gastou mais tempo. Dentro de cada linguagem em cada máquina, encontraremos estas facilidades, que devem ser exploradas e usadas, pois poderão ocorrer condições de erro de programa, que estas facilidades poderão auxiliar bastante na detecção dos mesmos.

CONSIDERAÇÕES FINAIS

E natural que a descoberta da necessidade de se utilizar de uma metodologia de desenvolvimento e testes de programas estruturados, venha a ocorrer quando a manutenção destes começa a crescer e tornar-se crítica, e , numa análise mais apurada das causas, concluímos que esta deve-se em parte ao fato dos programas terem sido mal desenvolvidos e testados.

Efetivamente, pode parecer a principio que este processo vem acarretar um custo maior no desenvolvimento, porque novos procedimentos estarão sendo acrescentados à atividade de programação. Por outro lado há de se observar que grande esforço empregado terá seu retorno já no desenvolvimento do próprio programa, pela produtividade maior que será alcançada na aplicação de um método objetivo.

Um outro aspecto a ser considerado é que o tempo decorrido para o desenvolvimento completo de um sistema deve diminuir sensivelmente porque para se obter o mesmo resultado em termos de qualidade e confiabilidade, através dos processos tradicionais, serão consumidos maiores recursos; nota-se que devemos considerar como recursos não só pessoal, como também equipamentos e materiais utilizados para desenvolver e testar os programas.

O retorno maior, porém, está localizado na fase de operação no sistema, onde teremos uma grande redução no tocante à ocorrência de problemas imprevistos.

Implantar esta metodologia, não é tarefa fácil, pois a adaptação do pessoal para a aplicação do método exige um trabalho de conscientização , com um alto grau de envolvimento, participação e motivação, no sentido de neutralizar as reações contrárias que surgem naturalmente quando ocorrem mudanças em uma organização.

B I B L I O G R A F I A

. METODOLOGIA CELEPAR - MC-DMS - CELEPAR

. MANUAL DE NORMAS E PROCEDIMENTOS DE PROGRAMAÇÃO CELEPAR

. A METODOLOGIA DE PROGRAMAÇÃO - JOSÉ DIDYK JUNIOR