FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

CLAUDETE WERNER

AVALIAÇÃO EXPERIMENTAL DO CRITÉRIO DE TESTE MUTAÇÃO DUAL

MARÍLIA - SP
2005

CLAUDETE WERNER

AVALIAÇÃO EXPERIMENTAL DO CRITÉRIO DE TESTE MUTAÇÃO DUAL

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Márcio Eduardo Delamaro

MARÍLIA
2005

WERNER, Claudete
Avaliação Experimental do Critério de Teste Mutação Dual / Claudete Werner; Orientador: Márcio Eduardo Delamaro.
Marília, SP: [s.n.], 2005.
157 f.

Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha.

1. Critério de Teste 2. Mutação Dual

CDD: 005.14

CLAUDETE WERNER

AVALIAÇÃO EXPERIMENTAL DO CRITÉRIO DE TESTE MUTAÇÃO DUAL

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM,/F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação.

Resultado: Aprovada

ORIENTADOR: Prof. Dr. Márcio Eduardo Delmaro

1º EXAMINADOR: Profa. Dra Maria Istela Cagnin

2º EXAMINADOR: Profa. Dra Sandra Camargo Pinto Ferraz Fabbri

Marília, 21 de Outubro de 2005.

Aos meus pais,

Maria Nelsi Werner

Cláudio José Werner, com saudades.

AGRADECIMENTOS

A Deus e à Nossa Senhora Aparecida,

Foi em quem, em momentos de desespero, encontrei forças para prosseguir e superar tantos obstáculos. Agradeço essa grande conquista profissional e principalmente pessoal.

Ao meu orientador, Prof. Dr. Marcio Eduardo Delamaro,

Pela confiança demonstrada, pelo apoio e pelos direcionamentos sempre oportunos na orientação deste trabalho.

À minha querida família,

Em especial à minha mãe, que, com sua sabedoria, me aconselhou e incentivou.
Ao meu pai (in memoriam), pela sua luz que ilumina o meu caminho.

Ao Prof. Dr. Edmundo Sergio Spotto,

Pela amizade, pelo companherismo e pelos incentivos em momentos difíceis.

À Fundação Ensino Eurípides Soares da Rocha,

Por oferecer o Mestrado em Computação, e onde tenho orgulho de conquistar esse titulo de mestre.

À Unipar - Universidade Paranaense,

Em especial à Diretora Geral da Unipar *Campus* Paranavaí, Prof. Dra Edwirge Viera Franco, pelo respeito, pela compreensão, e pela confiança depositados em nosso trabalho.

Aos meus amigos do mestrado, que fizeram muita diferença em minha vida ao longo desses dois anos e meio. Sem eles tudo seria bem mais difícil (Gisele, Montanha, Afonso, Junior, Ricardo, Luiz Fernando entre muitos outros).
Ao aluno da inciação cientifica Gustavo Moraes Habermann, pela ajuda na realização dos experimentos.

"O SENHOR é o meu pastor, nada me faltará.
Deitar-me faz em verdes pastos, guia-me mansamente a águas tranqüilas.
Refrigera a minha alma; guia-me pelas veredas da justiça, por amor do seu nome.
Ainda que eu andasse pelo vale da sombra da morte, não temeria mal algum, porque tu estás comigo; a tua vara e o teu cajado me consolam.
Preparas uma mesa perante mim na presença dos meus inimigos, unges a minha cabeça com óleo, o meu cálice transborda.
Certamente que a bondade e a misericórdia me seguirão todos os dias da minha vida; e habitarei na casa do SENHOR por longos dias."

SALMOS 23

WERNER, Claudete. **Avaliação experimental do critério de teste mutação dual**. 2005. 157 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

A crescente demanda por sistemas baseados em computação e a exigência de softwares de maior qualidade vêm provocando um interesse pelas atividades agregadas aos projetos de Teste de Software. Vários estudos desenvolvidos nessa área comprovaram que o Teste de Software é um elemento crítico da garantia de qualidade e representa a revisão final da especificação, projeto e geração de código. Nessa perspectiva, diversas técnicas, critérios e ferramentas de teste têm sido propostas. Dos critérios de testes pesquisados, alguns se destacam como é o caso do critério Análise de Mutantes, por apresentar uma maior flexibilidade e eficácia que os demais. O presente trabalho está inserido nesse contexto e teve como objetivo a realização de estudos empíricos para avaliar o Critério de Teste de Mutação Dual, procurando definir uma estratégia eficaz e ao mesmo tempo de baixo custo para realização de testes. O critério de Teste de Mutação Dual foi comparado com o Critério de Mutação, por meio de ánalises estatísticas dos resultados dos experimentos. Comprovou-se que existe diferença significativa entre os dois critérios, apresentando resultados favoráveis ao Critério Dual.

Palavras-chave: Teste de Software, Critérios de Teste, Análise de Mutantes, Mutação Dual.

WERNER, Claudete. **Avaliação experimental do critério de teste mutação dual**. 2005. 157 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The great demand for computation systems as well as the requirement for better software has incresead the interest for software test design. Some studies developed on this area proved that software testing is a critical element for quality control and it also performs the final revision of specification, design and code generation. In this sense many techiniques, criteria and tools have been proposed. Among the investigated testing criteria, the Mutation Analysis is a very relevant one, as it is more flexible and efficient than the others. This research has the objective of performing empirical studies in order to evaluate the Dual Mutation Testing Criterion, trying to define an efficient strategy that could also have low costs for test performing. the Dual Mutation Testing Criterion was compared to the Mutation Criterion by means of statistical analysis of the results and experiments. We were able to prove that there is a significant difference between the two criteria, as the Dual Mutation Testing Criterion had better results than the Mutation Criterion.

Key Words: Softwares Testing, Testing criteria, Mutant Analysis, Dual Mutation.

LISTA DE TABELAS

LISTA DE FIGURA

SUMÁRIO

1 INTRODUÇÃO

A Engenharia de Software surgiu na década de 70 e vem evoluindo significativamente nas ultimas décadas, procurando estabelecer técnicas, critérios, métodos e ferramentas para a produção de software. Em conseqüência da crescente utilização de sistemas baseados em computação em praticamente todas as áreas da atividade humana, há um aumento na demanda por qualidade e produtividade, tanto do ponto de vista de produção como do ponto de vista dos produtos gerados (DÓRIA, 2001).

Mesmo com o uso de métodos, técnicas e ferramentas nas diversas fases de produção do software, alguns erros podem ainda permanecer no produto. Para que o produto de software atinja um grau de qualidade aceitável, são necessárias atividades de garantia de qualidade, dentre elas, de verificação, validação e teste (VV&T). Dentre essas técnicas, as de teste de software são as mais utilizadas. Elas têm por objetivo projetar uma série de casos de teste nos quais há uma grande probabilidade de serem encontrados erros. Essas técnicas fornecem diretrizes sistemáticas para projetar testes que exercitam a lógica interna dos componentes de software e que exercitam também os domínios de entrada e de saída do programa para descobrir erros na função, no comportamento e no desempenho do programa (PRESSMAN, 2002).

Apesar de não ser possível, por meio de testes, provar que um programa está correto, os testes, se conduzidos sistemática e criteriosamente, contribuem para aumentar a confiança de que o software desempenha as funções especificadas e contribuem também para evidenciar algumas características mínimas do ponto de vista da qualidade do produto (DÓRIA, 2001).

A atividade de teste de software pode ser dividida em três perspectivas diferentes:

(1) A lógica interna do programa é exercitada usando técnicas de projeto de casos de teste do tipo **Caixa Branca ou Estrutural**.

(2) Requisitos de software são exercitados usando-se técnicas de projeto de casos de teste do tipo **Caixa-Preta ou Funcional**.

(3) O **Teste Baseado em Defeitos** utiliza informações sobre os erros mais freqüentes cometidos durante o processo de desenvolvimento do software e sobre os tipos específicos de erros que deseja revelar.

Em todos os casos, o objetivo é encontrar o maior número de erros com a menor quantidade de esforço e de tempo.

Neste capítulo, discutimos inicialmente o objetivo e a motivação para a realização desta pesquisa e, ao final, apresentamos a forma de organização deste trabalho.

1.1 Objetivo

Várias pesquisas e estudos empíricos vêm sendo desenvolvidos na Engenharia de Software. Na área de Teste de Software mais especificamente, existe um significativo esforço no desenvolvimento de estudos empíricos objetivando investigar quais técnicas e critérios são mais adequados e efetivos, em determinadas circunstâncias, para descobrir erros e como essas técnicas podem ser utilizadas de maneira complementar. A diferença entre essas técnicas está na origem da informação que é utilizada para avaliar ou construir conjuntos de casos de teste, sendo que cada técnica possui uma variedade de critérios para esse fim (JORGE & DELAMARO, 2001).

Assim sendo, o objetivo principal deste estudo é avaliar e comparar técnicas de critérios de teste de Análise de Mutantes e avaliar o critério de teste de Mutação Dual, por meio de estudo experimental.

Durante os experimentos será realizada uma análise comparativa com a finalidade de apresentar uma estratégia de teste que demonstre um baixo **custo** de aplicação, uma alta

eficácia em revelar a presença de erros e bons resultados na **dificuldade de satisfação** para adequação dos critérios de testes avaliados.

1.2 Motivação

Várias pesquisas têm sido desenvolvidas com o intuito de determinar qual técnica ou critério de teste é mais adequado e efetivo, em determinadas circunstâncias, para descobrir determinados conjuntos de erros. Essas pesquisas também visam, de forma mais ampla, verificar como as técnicas podem ser aplicadas de forma complementar para melhoria da qualidade de software. Para tanto é preciso propor critérios eficientes, construir ferramentas que suportem e que validem esses critérios por meio de estudos experimentais e teóricos.

1.3 Organização do trabalho

Neste capítulo foram apresentados o contexto no qual este trabalho está inserido, a motivação para realizá-lo e os objetivos a serem atingidos. No capítulo 2 apresentam-se definições e conceitos relacionados a Fundamentos do Teste de Software, enfocando-se os objetivos, as atividades de teste e as técnicas de teste de software. No capítulo 3 aborda-se a Análise de Mutantes com seus principais conceitos, vantagens e limitações. Ainda no capítulo 3 serão apresentados o conceito e a definição de Mutação Dual. No capítulo 4 apresentam-se definições sobre estudos empíricos, expondo-se comparações de estratégias e processo de experimentação. No capítulo 5 encontram-se a avaliação empírica, a descrição dos experimentos, dos resultados e da análise estatística. As conclusões do trabalho constituem o capítulo 6 e, na parte final do trabalho, indicam-se as referências que forneceram a base teórica deste projeto.

2 FUNDAMENTOS DO TESTE DE SOFTWARE

Neste capítulo, discutimos alguns conceitos sobre a atividade de testes de software, bem como os objetivos e os níveis de teste. Em seguida são apresentadas as fases de teste e as principais técnicas e critérios que podem ser utilizados em cada uma delas.

2.1 Considerações iniciais

O teste expõe uma anomalia interessante para o engenheiro de software. Durante as primeiras atividades de engenharia de software, o engenheiro tenta construir um software, partindo de um conceito abstrato até chegar a um produto tangível.O engenheiro cria uma série de casos de testes que são destinados a "demolir" o software que foi construído (PRESSMAN, 2002). De fato, o teste é um passo do processo de software que poderia ser visto como destrutivo em vez de construtivo.

Engenheiros de Software são, por natureza, pessoas construtivas. O teste exige que o desenvolvedor reveja noções pré-concebidas da correção do software recém-desenvolvido e supere um conflito de interesses que ocorre quando erros são descobertos (PRESSMAN, 2002).

2.1.1 Objetivos do teste

MYERS (1979) enumera algumas regras que podem servir como objetivos do teste:

(1) Teste é um processo de execução de um programa com a finalidade de encontrar um erro.

(2) Um bom caso de teste é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto.

(3) Um teste bem-sucedido é aquele que descobre um erro ainda não descoberto.

Esses objetivos implicam em uma dramática mudança de ponto de vista. Eles vão contra a visão comum de que um teste bem-sucedido é aquele no qual não são encontrados erros. O real objetivo deve ser projetar testes que descubram sistematicamente diferentes classes de erros e façam-no com uma quantidade mínima de tempo e esforço (PRESSMAN, 2002).

Se o teste for conduzido de maneira bem sucedida (de acordo com os objetivos declarados anteriormente), ele descobrirá erros no software. Como benefício secundário, o teste demonstra que as funções do software parecem estar funcionando de acordo com a especificação e que os requisitos de comportamento e de desempenho parecem estar sendo satisfeitos. Além disso, os dados coletados à medida que o teste é conduzido fornecem uma boa indicação da confiabilidade do software e alguma indicação da qualidade de todo o software. O teste não pode mostrar a ausência de erros e defeitos, mas apenas indicar que defeitos estão presentes (PRESSMAN, 2002).

O ideal deveria ser executar o programa sob teste com todos os valores possíveis do domínio de entrada. Sabe-se, entretanto, que o teste exaustivo é impraticável devido às restrições de tempo e de custo para realizá-lo. Dessa forma, é necessário determinar quais casos de teste utilizar, de modo que a maioria dos erros existentes possa ser encontrada e que o número de casos de teste e o tempo requerido não sejam tão grandes a ponto de o teste ser impraticável.

Por meio dos testes não é possível provar que um programa está correto. Os testes, se conduzidos com critério, contribuem para aumentar a confiança de que o software desempenha as funções especificadas e para apresentar algumas características mínimas do ponto de vista da qualidade do produto (CHAIM, MALDONADO & JINO, 2002).

Técnicas e critérios de teste têm sido elaborados com o objetivo de fornecer uma maneira rigorosa e sistemática para selecionar um subconjunto do domínio de entrada e, ainda assim, serem eficientes para apresentar os erros existentes, preocupando-se com o tempo e com o custo associados à atividade de teste.

2.2 A atividade de teste

Um software bem sucedido é o resultado de uma estratégia de teste de software que integre métodos de projeto de casos de teste em uma seqüência bem planejada de passos. A estratégia apresenta um guia que fornece os passos a serem conduzidos. Esses passos incluem as tarefas do testador, quando esses passos são planejados e depois executados e a quantidade de esforço, tempo e recursos necessários. Apesar dos métodos, das técnicas e das ferramentas empregadas nesse roteiro de desenvolvimento de software, erros no produto ainda podem ocorrer.

Nessa perspectiva a atividade de teste é de fundamental importância para a identificação e para a eliminação de erros que persistem, representando a última revisão da especificação, projeto e codificação (MALDONADO, 1991).

Os testes podem ser conduzidos em três níveis (PRESSMAN, 2002):

- **Teste de Unidade**: concentra esforços na menor unidade do projeto de software (módulo), ou seja, procura identificar erros de lógica e de implementação em cada módulo do software separadamente.
- **Teste de Integração**: é uma técnica sistemática para a construção da estrutura de programa, realizando, ao mesmo tempo, teste para descobrir erros associados às interfaces. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

- **Teste de Sistema**: é, na verdade, uma série de diferentes testes cujo objetivo é verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas.

Uma estratégia para teste de software pode ser vista no contexto espiral. O Teste de Unidade começa no centro da espiral e se concentra em cada unidade do software como implementado no código fonte. O teste progride, movendo-se para fora ao longo da espiral para o teste de integração, em que o foco fica no projeto e na construção da arquitetura do software.

Finalmente, chegamos ao teste do sistema, em que o software e os outros elementos do sistema são testados como um todo. Para testar o software de computador, movemos pela espiral para fora ao longo de voltas que ampliam o escopo do teste a cada volta (PRESSMAN, 2002).

A atividade de teste contribui no sentido de aumentar a confiança de que o software executa corretamente as funções especificadas. Em geral, não se pode provar que um programa está correto.

Apesar das limitações próprias da atividade de teste, sua aplicação de maneira sistemática e bem planejada pode garantir ao software algumas características mínimas que são importantes tanto para o estabelecimento da qualidade do produto quanto para o seu processo de evolução (DELAMARO, 1993). Assim, qualquer estratégia de teste deve incorporar planejamento de teste, projeto de casos de teste, execução de teste e a resultante coleta e avaliação de dados.

2.3 Técnicas de teste de software

Uma rica variedade de técnicas de projeto de casos de teste tem sido desenvolvida para o software. Essas técnicas fornecem ao desenvolvedor uma abordagem sistemática do

teste. Mais importante, as técnicas fornecem um mecanismo que pode ajudar na totalidade dos testes e ainda proporcionam uma probabilidade mais alta para descobrir erros no software.

As técnicas de teste podem ser classificadas em três grandes categorias:

- Técnica Funcional - Teste de Caixa Preta
- Técnica Estrutural - Teste de Caixa Branca
- Teste Baseado em Defeitos

O que diferencia uma técnica da outra é a origem da informação utilizada para avaliar ou construir os conjuntos de casos de teste, sendo que cada técnica possui uma variedade de critérios para esse fim (MALDONADO, 1991).

Na técnica funcional utiliza-se a especificação para derivar os requisitos de teste; na técnica estrutural os requisitos são derivados a partir dos aspectos de implementação do software; na técnica baseada em defeitos os elementos requeridos para caracterizar a atividade de teste são baseados nos defeitos mais comuns que o programador pode cometer durante o desenvolvimento do software.

Nenhuma dessas técnicas é suficientemente completa para garantir a qualidade da atividade de teste. Para obter sucesso, é necessário a combinação das três técnicas para garantir um teste de boa qualidade.

Segundo **Wong, Mathur e Maldonado (1994)**, alguns fatores básicos são necessários para comparar a adequação dos critérios de teste:

- **Custo:** esforço necessário para que o critério seja usado. Pode ser medido através do número de casos de testes requeridos para satisfazer o critério ou por outras métricas dependentes do critério, tais como o tempo necessário para executar todos os mutantes gerados, ou o tempo gasto para identificar os mutantes equivalentes, caminhos e associações não executáveis, construir manualmente os casos de teste e aprender a utilizar as ferramentas de teste.

- **Eficácia:** capacidade que um critério possui, em relação a outro critério, de detectar um maior número de erros.
- **Dificuldade de Satisfação:** probabilidade de satisfazer um critério tendo satisfeito outro. Seu objetivo é verificar o quanto se consegue satisfazer um critério C_1 tendo satisfeito um critério C_2 (C_1 e C_2 são incomparáveis, ou C_1 inclui C_2).

2.3.1 Técnica Funcional

A Técnica Funcional, (ou teste caixa-preta) é assim conhecida pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída (MYERS, 1979). Essa técnica focaliza os requisitos funcionais do software, isto é, o teste caixa-preta permite ao engenheiro de software derivar conjuntos de condições de entrada que devem exercitar plenamente todos os requisitos funcionais de um programa.

Apesar de serem projetados para descobrir erros, testes de caixa-preta são usados para demonstrar que as funções do software são operacionais, que a entrada é adequadamente aceita e a saída é corretamente produzida, e que a integridade da informação externa é mantida. Um teste de caixa-preta examina algum aspecto fundamental do sistema e se preocupa pouco com a estrutura lógica interna do software.

O teste caixa-preta tenta encontrar erros das seguintes categorias (PRESSMAN, 2002):

1 Funções incorretas ou omitidas;

2 Erros de interface;

3 Erros de estrutura de dados ou de acesso à base de dados externa;

4 Erros de comportamento ou de desempenho;

5 Erros de iniciação e término.

Exemplos de critérios de teste funcional são (PRESSMAN, 2002):

- **Particionamento em Classes de Equivalência**: a partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada de um programa em classes de equivalência válidas e inválidas. Em seguida seleciona-se o menor número possível de casos de teste, baseando-se na hipótese de que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválidas deve ser gerado um caso de teste distinto. O uso de particionamento permite examinar os requisitos mais sistematicamente e restringir o número de casos de teste existentes. Alguns autores também consideram o domínio de saída do programa para estabelecer as classes de equivalência.
- **Análise do Valor Limite**: é um complemento ao critério de Particionamento em Classes de Equivalência, sendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa; ao invés de selecionar-se qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois nesses pontos concentra-se um grande número de erros. O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.
- **Grafo de Causa-Efeito**: os critérios anteriores não exploram combinações das condições de entrada. Esse critério estabelece requisitos de teste baseados nas possíveis combinações das condições de entrada. Primeiramente são levantadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) do programa. A seguir é construído um grafo relacionando as causas e efeitos levantados. Esse grafo é convertido em uma tabela de decisão a partir da qual são derivados os casos de teste.

2.3.2 Técnica Estrutural

A técnica de teste caixa-branca (ou teste estrutural) é um método de projeto de casos de teste que usa a estrutura de controle de projeto procedimental para derivar casos de teste.

Os testes de estrutura são, em geral, aplicados a unidades de programa relativamente pequenas, como subrotinas, ou às operações associadas com um objeto. O testador pode analisar o código e utilizar conhecimentos sobre a estrutura de um componente a fim de derivar os dados para o teste. A análise do código pode ser utilizada para descobrir quantos casos de teste são necessários para garantir que todos os requisitos sejam executados pelo menos uma vez durante o processo de teste (RAPPS & WEYUKER, 1985; FRANKL, 1987; HOWDEN, 1987; NTAFOS, 1988).

Na técnica de teste estrutural, os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural baseia-se no conhecimento da estrutura interna da implementação.

Conforme definições de PRESSMAN (2002), os critérios de teste estrutural são, em geral, classificados em:

1 **Critérios Baseados em Fluxo de Controle**: utilizam apenas características de controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos dessa classe são:

- **Todos-Nós**: exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez.
- **Todos-Arcos**: requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa seja exercitada pelo menos uma vez.
- **Todos-Caminhos**: requer que todos os caminhos possíveis do programa sejam executados.

2 **Critérios Baseados em Fluxo de Dados**: utilizam informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições para estabelecerem os requisitos de teste. Exemplos dessa classe de critérios são os **Critérios de Rapps e Weyuker** e os **Critérios Potenciais-Usos**.

3 **Critérios Baseados na Complexidade**: utilizam informações sobre a complexidade do programa para derivar os requisitos de teste. Um critério bastante conhecido dessa classe é o **Critério de McCabe**, que utiliza a complexidade ciclomática do grafo de programa para derivar os requisitos de teste. Essencialmente, esse critério requer que um conjunto de caminhos linearmente independentes do grafo de programa seja executado.

Um problema relacionado ao teste estrutural é a impossibilidade, em geral, de se determinar automaticamente se um caminho é ou não executável, ou seja, não existe um algoritmo que, dado um caminho completo qualquer, decida se o caminho é executável e forneça o conjunto de valores que causam a execução desse caminho (VERGÍLIO, MALDONADO & JINO, 1993).

2.3.3 Teste baseado em defeito

A técnica de teste baseada em defeito utiliza informações sobre os tipos de erros mais freqüentes no processo de desenvolvimento de software para derivar os requisitos de teste. A ênfase da técnica está nos erros que o programador pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência (MALDONADO *et al*., 1998).

Dois critérios típicos que se concentram em erros são os critérios de Semeadura de Erros (Error Seeding) (BUDD, 1981) e de Análise de Mutantes (Mutation Analysis) (DEMILLO, LIPTON & SAYWARD, 1978).

As técnicas baseadas em erros derivam casos de teste a partir de erros específicos (ou classes de erros) comuns em linguagens de programação. O objetivo é mostrar a presença ou ausência de tais erros no programa. O principal critério baseado em erros é a Análise de Mutantes.

No critério de Semeadura de Erros, uma quantidade conhecida de erros é semeada artificialmente no programa. Após o teste, do total de erros encontrados, verificam-se quais são naturais e quais são artificiais. Usando estimativas de probabilidade, o número de erros naturais ainda existentes no programa pode ser calculado (GOEL, 1985).

O Teste de Analise de Mutantes é um critério que se baseia nos erros que podem ser cometidos ao longo do processo de desenvolvimento do software. Para modelar esses erros, um conjunto de operadores de mutação é definido e aplicado ao produto em teste, gerando versões modificadas do produto, chamadas mutantes.

Basicamente, a aplicação desse critério envolve as seguintes atividades:

- execução do programa com um conjunto de casos de teste T;
- geração dos mutantes;
- execução dos mutantes com T;
- análise dos mutantes vivos.

Um programa P é testado com um conjunto de casos de teste T. Se o programa funciona corretamente, então P sofre pequenas perturbações, gerando mutantes que são executados com T. Caso o comportamento de P' (um mutante de P) seja diferente de P, então esse mutante é dito "morto". Caso contrário, esse mutante está "vivo" devido a um entre dois motivos:

1 O conjunto T não é suficiente (adequado) para distinguir o comportamento de P e P' e, com isso, novos casos de teste devem ser incluídos ao conjunto ou

2 P' é dito equivalente a P, ou seja, para qualquer dado do domínio de entrada o comportamento dos dois programas não difere.

A definição deste critério de teste fundamenta-se em duas hipóteses básicas (DEMILLO, LIPTON, SAYWARD, 1978):

- hipótese do programador competente: um programa construído por um programador competente está correto ou está próximo do correto. Os programas incorretos contêm um desvio sintático que leva a resultados errados, e
- hipótese do efeito de acoplamento: erros complexos são acoplados a erros simples de forma que casos de teste capazes de detectar erros simples são capazes também de detectar erros mais complexos.

A definição dos operadores de mutação é um ponto fundamental para aplicação do Teste de Mutação. Entende-se por operadores de mutação as regras que definem as alterações que devem ser aplicadas ao produto em teste, caracterizando um conjunto de implementações alternativas. Os operadores de mutação são construídos também para satisfazer a um entre dois propósitos:

1 induzir mudanças sintáticas simples com base nos erros típicos cometidos durante o processo de desenvolvimento (como trocar o nome de uma variável) ou

2 forçar determinados objetivos de teste (como executar todas as instruções de um programa).

De acordo com **Dória (2001)**, é importante ressaltar que as técnicas de teste devem ser vistas como complementares e a questão está em como utilizá-las de forma que as vantagens de cada uma sejam melhor exploradas em uma estratégia de teste que leve a uma

atividade de teste de boa qualidade, ou seja, eficaz e de baixo custo. As técnicas e critérios de teste fornecem ao desenvolvedor uma abordagem sistemática e teoricamente fundamentada, além de constituírem um mecanismo que pode auxiliar a avaliar a qualidade e a adequação da atividade de teste.

Considerando a quantidade de critérios de testes que vêm sendo pesquisados pelo meio científico, um fator relevante é a escolha e/ou a determinação de uma estratégia de teste, que, em última análise, passa pela escolha de critérios de teste, de forma que as vantagens de cada um desses critérios sejam combinadas objetivando uma atividade de teste de maior qualidade.

Estudos teóricos e empíricos de critérios de teste são de extrema relevância para a formação desse conhecimento, fornecendo subsídios para o estabelecimento de estratégias de baixo custo e alta eficácia. As vantagens e desvantagens dos critérios de teste também podem ser avaliadas por meio dos estudos teóricos e empíricos, conforme estudo feitos por Dória (2001).

2.4 Considerações Finais

Este capítulo apresentou uma revisão bibliográfica das principais características da atividade de teste, mostrando as técnicas e alguns critérios de teste mais utilizados.

A atividade de teste encontra-se fundamentada por vários critérios, apresentados com o objetivo de fornecer uma forma rigorosa para a elaboração de conjunto de casos de teste que sejam eficazes para detectar os erros existentes em um software.

No capítulo 3, serão apresentados com maior ênfase o critério de Análise de Mutantes e o Critério de Mutação Dual, com suas características e aplicações.

3 ANÁLISE DE MUTANTES

Neste capítulo será apresentada uma visão geral do Critério de Análise de Mutantes, com seus principais conceitos, vantagens e limitações. O critério de teste de Mutação Dual também será abordado neste capítulo.

3.1 Critério análise de mutantes

A Análise de Mutantes surgiu na década de 70 na Yale University e no Georgia Institute of Technology e tem se mostrado, por meio de trabalhos empíricos e teóricos, ser um critério atrativo para o teste de programas (DELAMARO & MALDONADO, 1993).

Um dos primeiros artigos que descrevem a idéia de mutantes foi publicado em 1978 (DEMILLO, LYPTON & SAYWARD, 1978). Os autores desse artigo baseiam-se na técnica conhecida como Hipótese do Programador Competente, na qual eles assumem que programadores experientes escrevem programas muito próximos do correto. Considerando a validade dessa hipótese, pode-se afirmar que erros são introduzidos nos programas por meio de pequenos desvios sintáticos que, embora não causem erros sintáticos, alteram a semântica do programa e, conseqüentemente, conduzem o programa a um comportamento incorreto. Para revelar tais erros, a Análise de Mutantes identifica os desvios sintáticos mais comuns e, por meio da aplicação de pequenas transformações sobre o programa de teste, encoraja o testador a construir casos de teste que mostrem que tais transformações levam a um programa incorreto (MALDONADO *et al*., 1998).

Para identificação e exclusão desses erros, é utilizado o critério de análise de mutantes que funciona da seguinte forma:

1 Deve-se a partir do programa P, gerar um conjunto de mutantes M;

2 Definir o conjunto de teste T e seus critérios;

3 Executar os mutantes;

4 Comparar os resultados obtidos com os resultados esperados.

Os resultados podem ser apresentados em 3 (três) formas:

- **Mutantes Mortos**: A morte do mutante ocorre quando o mutante é executado e seu comportamento é diferente de P. Isso significa que o mutante é sensível às diferenças entre P e M.
- **Mutantes Vivos:** Ao contrário dos mutantes mortos, se os comportamentos de P e M forem iguais, e não se consegue distinguir se entre P e M, diz-se então que o mutante está vivo.
- **Mutantes Equivalentes:** seu processo de identificação é o mesmo dos mutantes que são considerados vivos, mas quando um mutante é considerado equivalente, deve ser descartado do conjunto de teste.

É importante notar que tal procedimento exige, além da execução do programa sendo testado, que algumas tarefas sejam executadas de forma automática. Pela grande quantidade de processamento que demanda tarefas como gerar e executar os mutantes e comparar os resultados produzidos é praticamente impossível executá-las com precisão e eficiência sem o uso do computador. Assim, na prática, deve-se associar a aplicação do critério de um "sistema de mutação" que interage com o testador na execução do teste (DELAMARO & MALDONADO, 1993).

A análise de mutantes visa à seleção de casos de teste que sejam capazes de identificar a presença dos possíveis desvios sintáticos (defeitos) mais comuns. Para determinar um conjunto de casos de teste com essa propriedade, programas mutantes são gerados por meio da aplicação de operadores de mutação. Os operadores são regras que são aplicadas para definir alterações no programa em teste. Por exemplo, o operador eliminação de comando gera um programa mutante em que um comando do programa original foi

eliminado. Casos de teste devem então ser gerados para diferenciar o comportamento do programa original do comportamento dos programas mutantes. Quando existir essa diferença, o mutante é dito estar morto. Se a saída do programa original está correta, então ele está livre do possível defeito contido no programa mutante. O critério de análise de mutantes exige que todos os mutantes sejam mortos.

Analogamente aos critérios de teste estrutural, uma medida do grau de cobertura (chamada score de mutação) do conjunto de casos de teste T em relação ao critério análise de mutantes por meio de um operador de mutação θ é definido (DELAMARO & MALDONADO, 1993).

Com certeza o critério análise de mutantes vem ganhando muita popularidade no meio produtivo de software. Entretanto, existe uma desvantagem na sua utilização, já que, às vezes, a quantidade de mutantes gerados é muito grande e isso exige muito na hora da execução desses mutantes. Devido à sua complexidade computacional, essa atividade pode se tornar inviável.

Todavia existem estratégias que podem diminuir esses problemas. Essas estratégias podem ser divididas em duas partes:

- tentativa de diminuir o tempo de execução utilizando hardware com arquiteturas avançadas e/ou
- diminuir o número de mutantes gerados a partir de critérios alternativos derivados da análise de mutantes e diminuir também os conjuntos de casos de teste.

O critério de teste de análise de mutantes envolve a repetição de alguns passos básicos, que serão descritos a seguir.

Passos da Análise de Mutantes

Dado um programa P e um conjunto de casos de teste T cuja qualidade se deseja avaliar, pode-se aplicar o critério de Mutantes por meio dos seguintes passos (VINCENZI, 1998):

1) Execução do Programa

Nesse passo deve-se executar o programa original P para os casos de teste selecionados e verificar se o resultado é o esperado. Caso o programa apresente um comportamento diferente para algum caso de teste, então um erro foi descoberto e o processo termina. Caso nenhum dos testes revele erro, o teste continua no passo seguinte.

2) Geração de Mutantes

Os mutantes são gerados por meio da aplicação de operadores de mutação no programa P que está sendo testado. Os operadores de mutação são as regras que definem as alterações que devem ser aplicadas no programa original P. A aplicação de um operador de mutação gera, na maioria das vezes, mais que um mutante, pois, como P pode conter várias entidades que estão no domínio de um operador, então esse operador é aplicado a cada uma dessas entidades, uma de cada vez.

3) Execução dos Mutantes

Esse passo pode ser totalmente automatizado, não requerendo a intervenção humana. Cada um dos mutantes é executado usando os casos de teste de T. Se um mutante M apresenta resultados diferentes de P, isso significa que os casos de teste utilizados são sensíveis e conseguiram expor as diferenças entre P e M. Nesse caso, M está morto e é descartado. Por outro lado, se M apresenta comportamento igual a P, isso indica ou uma fraqueza em T, pois não conseguiu distinguir P de M, ou que P e M são equivalentes. Nesse caso, diz-se que M continua vivo (DELAMARO & MALDONADO, 1993).

Finalizada a execução dos mutantes pode-se calcular o **escore de mutação**, que relaciona o número de mutantes mortos com o número de mutantes gerados. A Análise de Mutantes fornece uma medida objetiva do nível de confiança da adequação dos casos de testes analisados. O escore de mutação varia no intervalo (0,1), sendo que quanto maior o escore, mais adequado é o conjunto de casos de teste para o programa sendo testado.

4) Análise dos Mutantes

Essa é a fase que requer mais intervenção humana. Primeiramente, é necessário analisar os mutantes que sobreviveram à execução com os casos de teste disponíveis e decidir se tais mutantes são equivalentes ou não ao programa original.

Se o escore de mutação já estiver bem próximo de 1 – o testador deve julgar o que é "bem próximo" - então o teste pode ser encerrado, e T pode ser considerado um bom conjunto de casos de teste para P. Caso se decida continuar o teste, é preciso analisar os mutantes que sobreviveram à execução com os casos de teste disponíveis e decidir se esses mutantes são ou não equivalentes ao programa original (DELAMARO & MALDONADO, 1993).

Uma vez decidido que um mutante não é equivalente ao programa P, deve-se, se o desejo é construir um conjunto de casos de teste adequado para P, elaborar um caso de teste que mate tal mutante. Em Demillo e Offutt (1991), é apresentada uma solução para se tentar criar automaticamente casos de teste que matem os mutantes sobreviventes. O método, chamado geração automática baseada em restrições, basicamente associa a cada operador de mutação um modelo que fornece, para cada mutante gerado por esse operador, uma restrição que deve ser satisfeita na construção dos casos de teste.

3.2 Teste de mutação dual

O teste de mutação dual é um critério de seleção de casos de teste. O objetivo do critério é selecionar aqueles mutantes que são mortos muito facilmente e usá-los para

construir casos de teste úteis, contribuindo para a confiabilidade da atividade de teste, levando-se em conta que o número de mutantes mortos é muito alto.

A proposta desse critério, segundo Delamaro (2004) é selecionar casos de teste que não distinguem mutantes, tendo sua notação apresentada da seguinte maneira: um caso de teste **t** tal que dado um mutante M_i, sendo $M_i(t) = P(t)$.

Intuitivamente, selecionar casos de teste dessa maneira levaria a subdomínios que negligenciariam a boa qualidade dos critérios de subdomínio de pelo menos uma maneira:

- A intersecção entre eles seria alta. Se forem escolhidos dois mutantes M_i e M_j e se é selecionado um caso de teste **t** que não executa a declaração mutada em M_i nem a declaração mutada em M_j, então é facilmente observado que **t** está na intersecção dos subdomínios correspondentes àqueles mutantes. Em geral, não seria difícil achar um caso de teste desse tipo (DELAMARO, 2004).

O que se deseja com o critério de teste de mutação dual é selecionar casos de teste que executam a declaração mutada e ainda assim produzem o mesmo comportamento do programa original.

Para esclarecer esse conceito, algumas definições são necessárias. Considere-se um programa sob teste **P**, um conjunto de teste **T**, um conjunto de mutantes **M**, e o domínio de *input D* de **P**. Então, de acordo com a técnica de teste de mutação dual, são três as definições (DELAMARO, 2004):

Definição 1

Um mutante $M_i \in \mathbf{M}$ é morto quando executado com **T,** se e somente se existe $\mathbf{t} \in \mathbf{T}$ e que satisfaça as duas condições:

- a execução de **P** com **t** executa o comando que foi mutado para criar M_i;

- $M_i(t) = P_i(t)$.

Essa definição estabelece as condições necessárias para se considerar um mutante morto. Na próxima definição veremos a definição de uma mutante dual-equivalente, antes devemos esclarecer que as notações $t \phi M_i$ e $t \phi Mi$ indicam respectivamente que o caso de teste t matou o mutante M_i e que o caso de teste t não matou o mutante Mi.

Definição 2

Um mutante $M_i \in M$ é dul equivalente a **P** se e somente se

$$\forall \mathrm{t} \in D, \mathbf{t} \phi M_i,$$

ou seja, um mutante é dual equivalente se o caso de teste t não execute a declaração mutada, porém faz o mutante ter o mesmo comportamento do programa original.

Definição 3

O escore da mutação dual do conjunto de teste **T** é dado por

$$ms(T,M,P) = \frac{\#\ de\ \ mutantes\ \ mortos}{|M|\ -\ \#\ \ de\ \ mutantes\ \ dual-equiv}$$

Essa definição é a mesma que já foi mudada na Definição 1.

Para não deixar dúvidas quanto aos critérios acima estudados, vamos observar o exemplo usado por Delamaro (2004) para demonstrar suas características. Nesse exemplo demonstrado na Figura 3.1 temos um programa com dois parâmetros inteiros de entrada (x e y), esses parâmetros podem ser maiores ou iguais a zero e foram usados para computar o valor de x^y como mostra o código do programa.

Na Tabela 3.1 temos três mutantes, gerados através de operadores de mutação implementados na ferramenta *PROTEUM/IM 2.0* mostrando como a mutação dual seleciona casos de testes específicos que seriam considerados inúteis na mutação tradicional. A próxima coluna mostra as condições requeridas como entrada para que ocorra a morte do mutante dual.

```
Int pow (int x, int y)
{
int s =1;
int i;
        for(i = 0; i < y; i++)
        {
                s *=x;
        }
        return s;
}
```

Figura 3.1 – Exemplo do programa que computa x^y . (Fonte: Delamaro, 2004)

Tabela 3.1 – Exemplo de mutantes duais e casos de testes que os matam. (Fonte: Delamaro, 2004)

Estado Original	Estado Mutando	Operador de Mutação	Valores para matar os mutantes
s * = x;	s * = 0;	CLSR	x = 0 y > 0
s * = x;	s * = 1;	VLSR	x = 1 y > 0
s * = x;	s * = 1;	CRCR	x = 1 y ≥ 2 y % 2 = 0
return s;	return – 1;	CRCR	Equivalentes

O ultimo mutante (ultima linha da Tabela 3.1) mostra que a mutação dual não esta livre dos mutantes equivalentes. Pois este é o um caso de teste que não faz o mutante retornar um valor igual ao programa original.

3.3 Considerações finais

Este capítulo apresentou uma visão geral do Critério de Teste de Análise de Mutantes e do Critério de Mutação Dual, mostrando suas principais características, suas vantagens e estratégias para diminuir os problemas dos critérios em questão. Características relacionadas à utilização desses critérios foram demonstradas, apresentando os passos que compõem a sua aplicação.

A Análise de Mutantes tem se mostrado bastante eficaz na detecção de erros, entretanto, pelo fato de gerar um grande número de mutantes, sua aplicação se torna restritiva com relação ao tempo de processamento e quanto aos custos.

No capítulo 4, serão apresentadas definições sobre Engenharia Experimental, por meio de alguns estudos experimentais fundamentados no critério de Análise de Fluxo de Dados e no critério baseado em Mutação.

4 ESTUDOS EMPÍRICOS

4.1 Considerações iniciais

Estudos empíricos são motivados pela necessidade de se saber avaliar as tecnologias que surgem a cada momento para construir uma base sólida de informações a fim de contribuir para a evolução dessas tecnologias e da área de computação como um todo. Avaliar o custo/beneficio das técnicas de Teste de Software tem sido uma preocupação constante dos pesquisadores nas últimas décadas (BARBOSA, VINCENZI & MALDONADO, 1998; BASILI & REITER, 1981; DELAMARO, 1997; DELAMARO, MALDONADO & MATHUR, 1998; DELAMARO *et al*., 1998).

Pesquisas em estudos empíricos procuram, por meio da comparação entre os critérios, obter uma estratégia que seja eficaz para revelar a presença de erros no programa, ao mesmo tempo em que apresente um baixo custo de aplicação.

Experimentos são normalmente feitos em ambientes de laboratório, que fornecem um alto nível de controle. Quando os sujeitos do experimento são designados a diferentes tratamentos aleatoriamente, o objetivo é manipular uma ou mais variáveis e controlar todas as outras variáveis em níveis fixos. O efeito da manipulação é medido e, com base nisso, uma análise estatística pode ser realizada. Um exemplo de um experimento em engenharia de software é comparar dois diferentes métodos para inspeções. Para esse tipo de estudo, métodos de inferência estatística são aplicados com o propósito de mostrar com significância estatística que um método é melhor do que o outro (MONTGOMERY, 1997; SIEGEL & CASTELLAN, 1988; ROBSON, 1993).

Entrevistas são muito comuns nas ciências sociais em que, por exemplo, atitudes são coletadas (pesquisadas) para determinar como uma população irá votar na próxima eleição. Uma entrevista não fornece controle da execução da medição, embora seja possível comparar

com similares, mas é impossível manipular variáveis como em outros métodos de investigação (BABBIE, 1990).

Pesquisa de estudo de caso é uma técnica na qual fatores chave que podem ter efeito sobre o resultado são identificados e então a atividade é documentada (YIN, 1994; STALE, 1995).

Pesquisa de estudo de caso é um método observacional, ou seja, é feito pela observação de uma atividade ou de um projeto em andamento.

Um experimento é uma investigação formal, rigorosa e controlada. Em um experimento os fatores chave são identificados e manipulados. A separação entre estudo de caso e experimento pode ser representada pela noção de uma variável de estado (PFLEEGER, 1994-1995).

Em um experimento, a variável de estado pode assumir valores diferentes e o objetivo normalmente é distinguir entre duas situações, por exemplo, uma situação de controle e a situação sob investigação.

Algumas das estratégias de pesquisa poderiam ser classificadas tanto como qualitativas como quantitativas, dependendo do planejamento da investigação (Tabela 4.1). A classificação de uma entrevista depende do planejamento dos questionários, por exemplo, que dados são coletados e se é possível aplicar algum método estatístico. Isso também é verdadeiro para estudos de caso, mas a diferença é que uma entrevista é feita em retrospectiva ao passo que um estudo de caso é feito enquanto um projeto é executado.

Experimentos são puramente quantitativos uma vez que eles têm o foco em medir as diferentes variáveis, substituí-las e então medi-las novamente. Durante essas investigações, dados quantitativos são coletados e então métodos estatísticos são aplicados.

4.2 Características de experimentos

Experimentos são apropriados para investigar diferentes aspectos incluindo:

- Confirmar teorias, por exemplo, testar teorias existentes.
- Confirmar conhecimento convencional, por exemplo, testar as concepções das pessoas.
- Explorar relacionamentos, por exemplo, testar se uma certa relação se estabelece.
- Avaliar a precisão de modelos, por exemplo, testar se a precisão de certos modelos é a esperada.
- Validar medidas, por exemplo, certificar que uma medida realmente meça o que deve medir.

A força de um experimento é que ele possa investigar em que situações as declarações são verdadeiras e podem fornecer um contexto em que certos padrões, métodos e ferramentas são recomendados para uso.

Tabela 4.1 – Qualitativo vs. quantitativo em estratégias empíricas (Fonte: Wohlin,C. *et al*.,2002)

Estratégia	**Qualitativo / Quantitativo**
Entrevista	Ambos
Estudo de caso	Ambos
Experimento	Quantitativo

4.3 Processo de experimento

Desenvolver um experimento envolve cinco fases gerais. Os passos são:

- Definição
- Planejamento
- Operação
- Análise e interpretação

- Apresentação e empacotamento

A **definição** é o primeiro passo, no qual definimos o experimento a partir do problema, do objetivo e dos alvos. Em seguida, tem-se o **planejamento**, em que o projeto do experimento é determinado, a instrumentação é considerada e as ameaças ao experimento são avaliadas. A **operação** do experimento segue-se ao projeto. Na fase operacional, medidas são coletadas, analisadas e avaliadas na fase de **análise** e **interpretação**. Finalmente, os resultados são apresentados e empacotados na etapa da **apresentação** e **empacotamento (agrupamento)**. O processo não é para ser considerado um "modelo de cascata" verdadeiro, pois não se assume que uma atividade acaba quando a atividade seguinte se inicia. A ordem das atividades no processo indica primariamente a ordem de início das atividades. Em outras palavras, o processo é interativo e pode ser necessário voltar e refinar uma atividade anterior antes de continuar o experimento. A principal exceção é quando o experimento já começou, então é impossível voltar à definição e ao planejamento do experimento. Isso não é possível uma vez que iniciar a operação significa que os sujeitos estão influenciados pelo experimento e, se voltarmos, há o risco de ser impossível usar os mesmos sujeitos quando retornarmos à fase da operação no processo de experimento.

Conforme Wohlin *et al.* (2002), apresentamos as definições das etapas do processo de um experimento.

Definição

A primeira atividade é a definição. A hipótese deve ser declarada claramente e não precisa ser declarada formalmente nesse estágio. Além disso, o objetivo e os alvos do experimento devem ser definidos. O alvo é formulado a partir do problema a ser resolvido.

Planejamento

A atividade de planejamento é a etapa na qual o projeto do experimento é determinado, a instrumentação é considerada e os aspectos da validade do experimento são avaliados. Nessa etapa, a hipótese do experimento é declarada formalmente, incluindo uma hipótese nula e uma hipótese alternativa.

O próximo passo da atividade de planejamento é determinar variáveis, variáveis independentes (*inputs*) e variáveis dependentes (*outputs*). No que concerne às variáveis, é importante determinar os valores que as variáveis realmente podem assumir. Isso também inclui a determinação das escalas de medidas, que impõem restrições ao método que poderemos aplicar mais tarde para a análise estatística. Os sujeitos do estudo são identificados.

Um assunto relacionado ao projeto é a preparação para a instrumentação do experimento. Devemos identificar e preparar objetos adequados, desenvolver linhas de direção se necessário e definir procedimentos de medição.

Finalmente, é importante considerar a questão da variedade dos resultados que podemos esperar. A validade pode ser dividida em quatro classes principais: interna, externa, construção e validação da conclusão. Validade interna está relacionada com a validade no ambiente dado e a confiança nos resultados. A validade externa é uma questão de quão gerais são as descobertas. Muitas vezes gostaríamos de declarar que os resultados de um experimento são válidos fora do contexto ideal no qual o experimento foi realizado. A validade de construção é uma questão de julgar se o tratamento reflete uma construção da causa e se o resultado fornece uma figura verdadeira da construção do efeito. A validade de conclusão está relacionada com a relação entre o tratamento e o resultado do experimento. Temos que julgar se há uma relação entre o tratamento e o resultado.

Análise e interpretação

Os dados coletados durante a operação fornecem o input para essa atividade. Os dados agora podem ser analisados e interpretados. O primeiro passo na análise é tentar entender os dados usando estatística descritiva. Ela fornece uma visualização dos dados. A estatística descritiva nos ajuda a entender e interpretar os dados informalmente.

O próximo passo é considerar se o grupo de dados deve ser reduzido, seja por remover pontos de dados ou por reduzir o número de variáveis, estudando se algumas das variáveis fornecem as mesmas informações.

Após ter removido os pontos de dados ou reduzido os grupos de dados, podemos realizar um teste de hipótese em que o teste real é escolhido baseado em escalas de medição, valores nos dados de input e os tipos de resultados que estamos procurando. Um aspecto importante dessa atividade é a interpretação. Temos que determinar, a partir da análise, se a hipótese foi aceita ou rejeitada. Isso forma a base para a tomada de decisões e conclusões relativas a como usar os resultados do experimento, o que inclui motivação para estudos posteriores, por exemplo, para conduzir um experimento maior ou um estudo de caso. Dois métodos importantes relacionados à análise dos resultados são a triangulação e a meta-análise. A triangulação tem a ver com o emprego de múltiplos métodos para se obter um resultado confiável. A meta-análise é utilizada para resumir os resultados de vários estudos em uma única análise, por exemplo, combinar vários experimentos em uma análise.

Apresentação e empacotamento

Essa fase está relacionada com a apresentação e com o empacotamento dos resultados. Isso inclui primariamente a documentação dos resultados, o que pode ser feito por meio de um artigo de pesquisa para publicação, de um pacote de laboratório para propósitos de replicação ou como parte da base de experiência de uma companhia. Essa última atividade é importante para assegurar que as lições aprendidas sejam utilizadas para que seja dada uma devida atenção aos trabalhos futuros. Além disso, um experimento nunca irá fornecer a resposta final para uma questão. Assim, é importante facilitar a replicação do experimento. Uma documentação

compreensiva e completa é um pré-requisito para atingir esse objetivo. Dessa forma, devemos levar algum tempo após o experimento para documentá-lo e apresentá-lo de uma maneira apropriada.

4.4 Comparação de estratégias empíricas

Os pré-requisitos para uma investigação limitam a escolha da estratégia de pesquisa. Uma comparação de estratégias pode ser baseada em um número de diferentes fatores, como é mostrado na Tabela 4.2.

Tabela 4.2 – Fatores de estratégia de pesquisa (Fonte: Wohlin,C. *et al*.,2002)

Fator	Entrevista	Estudo de caso	Experimento
Controle de execução	não	sim	Sim
Controle de medida	não	não	Sim
Custo de investigação	baixo	médio	Alto
Facilidade de replicação (repetição)	alta	baixa	alta

O controle de execução descreve quanto controle o pesquisador tem sobre o estudo. Por exemplo, em um estudo de caso os dados são coletados durante a execução de um projeto. Se a gerência decide parar o projeto estudado devido a razões econômicas, por exemplo, o pesquisador não pode continuar o estudo de caso. O oposto é o experimento em que o pesquisador está no controle da execução.

Controle de medida é o grau de decisão segundo o qual o pesquisador pode decidir que medidas serão coletadas, incluídas ou excluídas durante a execução do estudo. Um exemplo é como coletar dados sobre volatilidade de exigência.

Relacionado aos fatores citados está o **custo de investigação**. Dependendo da estratégia escolhida, o custo difere. Isso está relacionado, por exemplo, ao tamanho da investigação e à necessidade de recursos. A estratégia com o custo mais baixo é a entrevista, uma vez que não requer uma grande quantidade de recursos. A diferença entre estudos de caso e experimentos é que, se escolhermos investigar um projeto em um estudo de caso, o resultado do projeto é alguma forma de produto que pode ser vendido no varejo, por exemplo,

é uma investigação on-line. Em um experimento off-line o resultado é alguma forma de experiência que não é rentável da mesma maneira que um produto.

Outro aspecto importante a considerar é a possibilidade de **replicar** a investigação. O propósito de uma replicação é mostrar que o resultado do experimento original é valido para uma população maior. A replicação se torna uma replicação "verdadeira" se é possível replicar tanto o formato (*design*) quanto os resultados. Não é incomum que o objetivo seja realizar a replicação, mas os resultados, em parte, saem diferentes dos resultados do estudo original.

A replicação de um experimento envolve repetir a investigação sob condições idênticas, mas com outra população. Isso ajuda a encontrar quanta confiança é possível colocar nos resultados de um experimento. Se a suposição da aleatorização estiver correta, isto é, se os sujeitos são representativos de uma população maior, replicações nessa população mostram os mesmos resultados do experimento realizado anteriormente. Se não tivermos os mesmos resultados, fomos incapazes de capturar todos os aspectos no *design* do experimento que afetam o resultado. Mesmo se for possível medir uma certa variável ou replicar um experimento, poderia ser difícil e muito custoso.

A escolha da estratégia empírica depende dos pré-requisitos da investigação, do seu propósito, dos recursos disponíveis e de como gostaríamos de analisar os dados coletados.

4.5 O Empirismo em um Contexto de Engenharia de Software

Por que deveríamos realizar engenharia de software empírica? As principais razões para realizar estudos empíricos quantitativos são a oportunidade de se obter resultados significativos e estatisticamente relativos ao controle, à predicação e à melhoria do desenvolvimento de software.

Para se obter sucesso no desenvolvimento de software, há algumas exigências básicas (BASILI, 1985; BASILI, 1989; DEMMING, 1986):

1 Compreensão do processo e do produto de software

2 Definição das qualidades do processo e do produto

3 Avaliação dos sucessos e das falhas

4 *Feedback* da informação para controle do projeto

5 Aprender a partir da experiência

6 Empacotamento e reuso da experiência relevante

Estudos empíricos são importantes para apoiar a realização dessas exigências e para o encaixe no contexto da pesquisa de engenharia de software industrial e acadêmica, assim como, em uma organização de aprendizado, buscar melhorias contínuas.

A seguir, são apresentados alguns experimentos já realizados na área de critérios de teste.

4.6 Experimentos baseados em critérios de análise de fluxo de dados e critérios baseados em mutação

Estudos empíricos foram conduzidos para a determinação da complexidade dos critérios baseados em análise de fluxo de dados e em critérios baseados em mutação, resultando em uma avaliação empírica desses e de outros critérios de teste, objetivando a determinação de um modelo para estimativa do número de casos de teste necessários.

Essa determinação é muito importante para as atividades de planejamento do desenvolvimento, por diversas razões. Weyuker (1990) caracterizou um *benchmark* para a avaliação empírica de uma família de critérios de teste estruturais. O mesmo *benchmark* foi aplicado para uma primeira avaliação empírica dos critérios Potenciais-Usos (VERGÍLIO, MALDONADO & JINO, 1992). Com a aplicação do *benchmark*, obtiveram-se resultados

bastante interessantes. Em geral, pode-se dizer que os critérios Potenciais-Usos, do ponto de vista prático, são factíveis e demandam um número de casos de teste relativamente pequeno.

Através de estudos empíricos têm-se obtido evidências de que a Análise de Mutantes também pode constituir na prática um critério atrativo para o teste de programas (MATHUR & WONG, 1993). Tais experimentos, além de mostrar como a Análise de Mutantes se relaciona com outros critérios de teste, buscam novas estratégias a fim de reduzir os custos associados ao critério.

Em outro trabalho realizado por Mathur e Wong (1994) foi comparada a adequação de conjuntos de casos de teste em relação aos critérios Análise de Mutantes e Todos-Usos. O objetivo do experimento foi verificar a dificuldade de satisfação entre os dois critérios, bem como seus custos, uma vez que esses critérios são incomparáveis do ponto de vista teórico. Nesse estudo, os conjuntos de casos de teste Análise de Mutantes-adequados também se mostraram Todos-Usos-adequados. No entanto, os conjuntos de casos de teste Todos-Usos-adequados não se mostraram adequados para o critério Análise de Mutantes em muitos dos casos. Esses resultados demonstram que é mais difícil satisfazer o critério Análise de Mutantes do que o critério Todos-Usos, salientando que Análise de Mutantes inclui Todos-Usos (MATHUR & WONG, 1994).

Wong, Mathur e Maldonado (1994) utilizaram a Mutação Aleatória (10%) e a Mutação Restrita para comparar o critério Análise de Mutantes com o critério Todos-Usos. O objetivo foi verificar o custo, a eficácia e a dificuldade de satisfação desses critérios. Os autores forneceram evidências de que os critérios Todos-Usos, Mutação Aleatória (10%) e Mutação Restrita representam, nessa ordem, o decréscimo do custo necessário para a aplicação do critério devido ao número de casos de teste requeridos, ou seja, o critério Todos-Usos requer mais casos de teste para ser satisfeito do que a Mutação Restrita. Em relação à eficácia para detectar erros, a ordem (do mais eficaz para o menos) é Mutação Restrita,

Todos-Usos e Mutação Aleatória. Observou-se, com isso, que examinar somente uma pequena porcentagem de mutantes pode ser uma abordagem útil na avaliação e construção de conjuntos de casos de teste. Desse modo, quando o testador possui pouco tempo para efetuar os testes, pode-se usar o critério de Análise de Mutantes para testar partes críticas do software, utilizando alternativas mais econômicas, tal como a Mutação Restrita ou o critério Todos-Usos para o teste das demais partes do software, sem comprometer significativamente a qualidade da atividade de teste.

Offutt, Tewary e Zhang (1996) também realizaram um experimento comparando o critério Análise de Mutantes com o critério Todos-Usos. Os resultados foram semelhantes àqueles obtidos por Wong, Mathur e Maldonado (1994), ou seja, o critério Análise de Mutantes revelou um maior número de erros do que o critério Todos-Usos e mais casos de testes foram necessários para satisfazer o critério Análise de Mutantes. Além disso, os conjuntos de casos de teste Análise de Mutantes-adequados foram adequados ao critério Todos-Usos, não sendo o inverso verdadeiro, resultado semelhante ao de Mathur e Wong (1994).

4.7 Considerações finais

Neste capítulo foram apresentados diversos estudos empíricos que comprovaram que o critério de Análise de Mutantes é altamente eficaz em revelar a presenças de erros (WONG, MATHUR & MALDONADO, 1994; OFFUTT, TEWARY & WANG, 1996; MATHUR & WONG, 1994; MATHUR & WONG, 1993). No entanto, problemas relacionados ao grande número de mutantes gerados tornam necessário o desenvolvimento de abordagens que permitam a sua utilização de forma economicamente viável.

No capítulo 5, serão apresentadas as avaliações experimentais do critério de teste de mutação dual, que é a proposta deste trabalho, bem como a metodologia utilizada e a análise estatística dos resultados.

5 AVALIAÇÃO EMPÍRICA

O objetivo deste capítulo é avaliar, por meio de estudos experimentais, a idéia de um novo critério de teste, intitulado “Teste de Mutação Dual” (DMT). O critério seleciona mutantes para criar casos de teste para um dado programa, similar ao que é feito no teste de mutação tradicional. A diferença é que esse critério considera mortos aqueles mutantes para os quais o testador forneceu um caso de teste que alcança o ponto de mutação e não distingue o mutante, isto é, que faz o mutante se comportar como o programa original (DELAMARO, 2004).

Ainda neste capitulo descreve-se a metodologia que foi utilizada para a realização dos experimentos.

5.1 Considerações iniciais

A literatura oferece algumas definições dos objetivos da experimentação em Engenharia de Software (Conradi et al., 2001).

Para compreender a natureza dos processos da informação, os pesquisadores devem observar o fenômeno, encontrar explicação, formular a teoria e verificá-la.

A experimentação pode ajudar a construir uma base de conhecimento confiável e reduzir incertezas sobre a adequação de teorias, ferramentas e metodologias.

> A observação e a experimentação podem levar a novos meios da introspecção, abrindo caminho para novas áreas de investigação. A experimentação pode encontrar novas áreas nas quais a engenharia age lentamente.

A experimentação pode acelerar o processo eliminando abordagens inúteis e suposições errôneas. A experimentação ajuda também a orientar a engenharia e a teoria nas direções promissoras de pesquisa.

5.2 Objetos de estudo

Uma das atividades mais importantes de um experimento em engenharia de software é a seleção dos programas utilizados, pois normalmente os resultados obtidos são relacionados às características dos mesmos.

Para a realização deste experimento, foi selecionado um grupo de 6 programas. Tais programas foram escritos na linguagem C.

Os programas utilizados para a realização dos experimentos são os seguintes:

1 CALCDATE;

2 CALEN;

3 CALEND;

4 CAL;

5 JDAY-JDATE;

6 DATEPLUS.

Esses programas estão relacionados à manipulação de datas e foram obtidos na Internet, no endereço eletrônico: http://ftp.unicamp.br/pub/unix-c/calendars/. Os programas são de domínio público. Todos aceitam apenas parâmetros de linha de comando, o que facilita a geração aleatória de casos de teste.

A descrição completa da funcionalidade de cada programa e o perfil operacional usado na geração dos casos de teste são apresentados nos apêndices A, B, C, D, E e F.

5.3 Descrição do experimento

Com os 6 programas selecionados, foram realizados 360 experimentos, dos quais são 30 sessões de teste para o critério Normal e 30 sessões de teste para o critério Dual para cada programa. Os experimentos foram realizadas em plataforma Linux/Fedora 3, utilizando a

ferramenta de teste *PROTEUM/IM 2.0* - **P**rogram **T**esting **U**sing **M**utants (DELAMARO *et al.*, 2000).

5.3.1 Seqüência utilizada para a realização do experimento

A seqüência utilizada para a realização do experimento é descrita a seguir: Identificação do perfil operacional de cada programa; estudo da linguagem em que os *scripts* para geração dos casos de testes foram criados: shell *script* e tcsh; criação dos *Scripts* para geração do Perfil Operacional; criação dos *scripts* que, a partir do perfil operacional, geram os casos de teste aleatoriamente. Para a apresentação dos resultados dos casos de teste foi criado o *script* para o cruzamento entre os dois critérios.

Os *Scripts* de geração de casos de teste servem para gerar casos de testes que irão ser importados pela ferramenta de teste *PROTEUM/IM 2.0* para matar os mutantes criados. Como uma das características mais importantes desses *scripts* é gerar os casos de testes aleatoriamente, foi utilizado um servidor de números, que é um programa que é deixado ativo e em *background* no sistema que quando chamado através de um comando próprio, retorna um numero aleatório que é maior ou igual a um valor piso e menor que um valor teto. Quando ativamos esse servidor de números utilizamos o seguinte comando: "./server<semente><porta>", onde a semente é o numero em que o servidor ira se basear para gerar o numero aleatório e a porta é onde esse servidor ficará ativo.

O *scripts* do cruzamento tem o objetivo de cruzar os casos de teste. É realizado utilizando os conjuntos de casos de teste do critério dual e executado no critério normal. Esse processo é repetido para o cruzamento utilizando o conjunto de casos de teste do critério normal e executado no critério dual.

5.3.2 Passos para a realização dos experimentos

Criou-se uma sessão raiz de teste na qual são gerados os mutantes. Os mutantes são gerados através de alterações da especificação original, que são feitos com base em um conjunto de operadores denominados operadores de mutação, sendo que cada operador pode estar associado a um tipo ou uma classe de erros que se pretende revelar.

Para gerar os mutantes de cada programa, utilizou-se uma lista de operadores pré-definida através da *PROTEUM/IM 2.0* (Apêndice G) da qual foram escolhidos 65 **operadores de mutação de unidade**. Não usamos o grupo todo de operadores, considerando que alguns mutantes instrumentados não fazem sentido para a mutação dual. Por exemplo, o operador STRP, através desse operador não há uma maneira pelo qual o ponto de mutação seja executado e o mutante se comporte como programa original. Outra razão é que alguns operadores podem mudar radicalmente a forma gráfica do fluxo de controle, a ferramenta não esta disponível para tomar tal decisão.

Os operadores de mutação definem alterações sintáticas a serem realizadas em um programa P, com base nas quais são criados seus mutantes. Para Jorge *et al.* (2001), a definição de operadores de mutação é fundamental para a aplicação de critérios baseados em mutação. São esses operadores que caracterizam o critério, estabelecendo os requisitos de teste e os mutantes a serem satisfeitos.

Em seguida são criadas duas sessões de testes para gerar e incluir 15.000 casos testes, uma em modo Normal e outra em modo Dual, realizadas automaticamente por meio de *scripts*, específicos para cada programa.

As sessões DUAL E NORMAL são avaliadas manualmente por meio da interface gráfica da ferramenta *PROTEUM/IM 2.0*. Nessa etapa são avaliados os mutantes que ainda estão vivos e novos casos de testes são incluídos manualmente para matar os mutantes

restantes ou identificá-los como equivalentes. Essa etapa inicial é muito importante, pois é quando são identificados os mutantes equivalentes.

Na seqüência, cria-se outra sessão denominada equivalentes, com subsessões denominadas Normal e Dual. Nessas subsessões são incluídos os casos de testes inseridos manualmente na etapa anterior. Concomitantemente insere-se a lista de mutantes equivalentes e o escore de mutação é zerado.

A partir da estrutura descrita, geram-se 30 sessões para o critério de teste Dual e 30 sessões de teste para o critério de teste Normal.

A repetição da execução desse experimento é necessária para a avaliação da **eficácia, custo** e a **dificuldade de satisfação** dos critérios de teste estudados. A geração de casos de testes para cada sessão tem sua origem por meio de sementes aleatórias.

Em seguida, realiza-se a avaliação de cada sessão individualmente para que, novamente, seja eliminado qualquer mutante não morto durante o processo anterior.

Por fim efetua-se o cruzamento entre os critérios de teste Normal e Dual para as 60 sessões. Esse cruzamento é realizado utilizando os conjuntos de casos de teste do critério Dual e executando no critério Normal. Esse processo é repetido para o cruzamento utilizando o conjunto de casos de teste do critério Normal e executando no critério Dual. Os cruzamentos são executados por meio de *scripts* para avaliar a **dificuldade de satisfação** de um critério em comparação ao outro.

5.3.3 Análise estatística dos resultados dos testes

Na estatística experimental, especificamente na análise de variância, os testes de hipóteses são amplamente utilizados para se obter uma conclusão a respeito das fontes de variação consideradas nos modelos lineares. Para isso, é comum a utilização de sistemas

estatísticos que fornecem análises de variância e estatística *F*, entre outras, para a tomada de decisões (NESI, 2002).

No experimento de comparação entre dois tipos de critérios de testes de mutação, utilizou-se o delineamento experimental em blocos casualizados completos, constituído de dois tratamentos para os casos de testes, dois tratamentos para os mutantes equivalentes e dois tratamentos para o escore de cruzamento dos conjuntos de casos de teste, com 30 sessões repetidas para cada critério.

Os tratamentos foram divididos em dois blocos, sendo o bloco 1 composto pelo resultado do critério do teste Normal e o bloco 2 composto pelo resultado do critério do teste Dual. O objetivo dessa análise estatística foi avaliar se o resultado do critério do teste Normal apresentava diferenças em relação ao resultado do critério do teste Dual.

Utilizando-se o software PROC GLM (General Linear Models) do SAS (Statistical Analysis System) e fez-se uma análise de variância convencional (PIMENTEL GOMES, 1990). Para tanto, adotou-se o modelo linear superparametrizado, caracterizado por:

$$y_{ij=}\mu + \tau_i + \varepsilon_{ij}$$

em que:

y_{ij} : valor observado para *i*-ésimo tratamento na *j*-ésima repetição;

μ : constante inerente a todas as observações;

τ_i : efeito do *i*-ésimo tratamento;

ε_{ij} : erro aleatório relativo à observação y_{ij} .

A análise de variância convencional testa a hipótese de que os tratamentos não diferem entre si. Os fatores avaliados resultaram em três hipóteses com 1 grau de liberdade cada. As hipóteses testadas foram as seguintes: número de casos de testes efetivos, número de mutantes equivalentes e escore de cruzamento.

Testes de hipóteses:

Hipótese 1. As hipóteses testadas para o critério de testes para os programas CALCDATE, CALEN, CALEND, CAL, JDAY e DATEPLUS do número de casos de testes efetivos têm a forma:

$$H_0 : \mu_1 = \mu_2,$$

$$H_1 : \mu_1 \neq \mu_2$$

em que:

μ_1 : média dos valores do número de casos de testes efetivos para o critério Normal;

μ_2 : média dos valores do número de casos de testes efetivos para o critério Dual.

Hipótese 2. Para o critério de testes do número de mutantes equivalentes para os programas CALCDATE, CALEN, CALEND, CAL, JDAY e DATEPLUS, as hipóteses testadas têm a forma:

$$H_0 : \mu_1 = \mu_2,$$

$$H_1 : \mu_1 \neq \mu_2$$

em que:

μ_1 : média dos valores do número de mutantes equivalentes para o critério Normal;

μ_2 : média dos valores do número de mutantes equivalentes para o critério Dual.

Hipótese 3. As hipóteses testadas para o escore de cruzamento para os programas CALCDATE, CALEN, CALEND, CAL, JDAY e DATEPLUS têm a forma:

$$H_0 : \mu_1 = \mu_2,$$

$$H_1 : \mu_1 \neq \mu_2$$

em que:

μ_1 : média dos valores do escore de cruzamento para o critério Normal;

μ_2 : média dos valores do escore de cruzamento para o critério Dual.

5.3.4 Teste de médias entre tratamentos

O objetivo de testar hipóteses sobre médias verdadeiras é avaliar certas afirmações feitas sobre essas hipóteses. A partir da afirmação de que as médias de dois tratamentos são iguais, faz-se o teste de comparação entre duas médias. Essa técnica pressupõe distribuição normal da amostra estatística. A suposição será válida se a distribuição da variável em estudo seguir uma distribuição normal e a amostragem for aleatória, e em geral, com amostra suficientemente grande ($n \geq 30$) (OGLIARI & ANDRADE, 2005).

5.4 Resultados obtidos

Na tabela 5.1 apresentam-se as informações referentes à aplicação dos critérios de teste de análise de mutação normal e de mutação dual. Essas informações são: número de mutantes gerados por meio de uma lista de operadores específica para o experimento, número de casos de testes efetivos, número de mutantes mortos e número de mutantes equivalentes.

Os resultados obtidos demonstram o grau de complexidade dos programas testados, considerando-se os critérios aplicados no experimento. Por meio desses resultados, podem ser feitas algumas observações.

Para o programa *Calcdate*, foram gerados 5.532 mutantes por meio da lista de operadores específica para o experimento. Com a aplicação do critério de teste normal foram mortos 4.818 mutantes, utilizando-se 75,33 casos de testes efetivos e 714 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério

dual, foram obtidos 5.255 mutantes mortos, utilizando-se 37,53 casos de testes efetivos e foram identificados 277 equivalentes.

Para o programa *Calen*, foram gerados 5.186 mutantes por meio da lista de operadores específica para o experimento. Por meio do critério de teste normal foram mortos 4.701 mutantes, utilizando-se 30,13 casos de testes efetivos e 485 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério dual, foram obtidos 5.176 mutantes mortos, aplicando-se 15 casos de testes efetivos e foram identificados 10 equivalentes.

Para o programa *Calend*, foram gerados 4.835 mutantes por meio da lista de operadores específica para o experimento. Por meio do critério de teste normal foram mortos 4.294 mutantes, utilizando-se 26,53 casos de testes efetivos e 541 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério dual, foram obtidos 4.449 mutantes mortos, aplicando-se 23,93 casos de testes efetivos e foram identificados 386 equivalentes.

Para o programa *Cal*, foram gerados 4.068 mutantes por meio da lista de operadores específica para o experimento. Por meio do critério de teste normal foram mortos 3.742 mutantes, utilizando-se 76,93 casos de testes efetivos e 326 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério dual, foram obtidos 3.896 mutantes mortos, aplicando-se 22,96 casos de testes efetivos e foram identificados 172 equivalentes.

Para o programa *Jday*, foram gerados 2.610 mutantes por meio da lista de operadores específica para o experimento. Por meio do critério de teste normal foram mortos 2.540 mutantes, utilizando-se 34,63 casos de testes efetivos e 70 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério dual, foram obtidos

2.576 mutantes mortos, aplicando-se 10,03 casos de testes efetivos e foram identificados 34 equivalentes.

Para o programa *Dateplus*, foram gerados 1.836 mutantes por meio da lista de operadores específica para o experimento. Por meio do critério de teste normal foram mortos 1.440 mutantes, utilizando-se 32,3 casos de testes efetivos e 396 mutantes foram identificados como equivalentes. Utilizando-se o mesmo programa, aplicando-se o critério dual, foram obtidos 1.445 mutantes mortos, aplicando-se 3 casos de testes efetivos e foram identificados 391 equivalentes.

Tabela 5.1: Resultados da avaliação dos Critérios de Teste Mutação e Mutação Dual

Programa	Número de mutantes						
	Gerados	Casos de Testes Efetivos		Mortos		Equivalentes	
		Normal	Dual	Normal	Dual	Normal	Dual
Calcdate	5.532	75,33	37,53	4.818	5.255	714	277
Calen	5.186	30,13	15	4.701	5.176	485	10
Calend	4.835	26,53	23,93	4.294	4.449	541	386
Cal	4.068	76,93	22,96	3.742	3.896	326	172
Jday	2.610	34,63	10,03	2.540	2.576	70	34
Dateplus	1.836	32,3	3	1.440	1.445	396	391

5.4.1 Escore de cruzamento do resultado dos experimentos entre os dois critérios

Na tabela 5.2 temos o escore de mutação do cruzamento entre os dois critérios. Foram utilizados os conjuntos de casos de testes do critério dual executado no critério normal e vice-versa.

Para o programa *Calcdate*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,817401. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,652488, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Para o programa *Calen*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,892673. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,519887, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Para o programa *Calend*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,884740. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,518610, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Para o programa *Cal*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,942834. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,445192, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Para o programa *Jday*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,959063. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,524340, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Para o programa *Dateplus*, os casos de testes usados na mutação dual, quando utilizados na mutação normal, obtiveram um escore de mutação de 0,784314. Já os casos de teste de critério normal, quando utilizados no critério dual, obtiveram um escore de 0,556609, demonstrando a maior adequação do critério dual sobre a critério normal, com relação à **eficácia,** ao **custo** e à **dificuldade de satisfação**.

Tabela 5.2: Escore de Mutação do cruzamento do resultado dos experimentos entre os dois critérios

Programas	Casos de Testes Efetivos		Escore de Cruzamento	
	Normal	Dual	Normal	Dual
Calcdate	32,3	61,13	0,817401	0,652488
Calen	12,7	14,23	0,892673	0,519887
Calend	22,4	25,06	0,884740	0,518610
Cal	14,53	43,8	0,942834	0,445192
Jday	9,23	24,9	0,959063	0,524340
Dateplus	3	8,56	0,784314	0,556609

O resultado do cruzamento entre os dois critérios apresentou melhor resultado do escore de mutação do cruzamento para o critério normal, em virtude da utilização do conjunto de casos de testes do critério dual. Esses resultados enaltecem o Critério de Mutação Dual, com relação à adequação do **custo,** da **eficácia e da dificuldade de satisfação**.

5.5 Análise estatística

Avaliando-se estatisticamente os resultados experimentais do presente trabalho (Apêndice H), utilizando nível de significância de $\alpha = 0,05$, ou seja, supondo como sendo de 5% o erro máximo das conclusões (OGLIARI & ANDRADE, 2005), rejeitou-se a hipótese *H0* para os três fatores avaliados de que as médias dos valores dos resultados dos testes Normal e Dual são iguais, como pode ser observado nas Tabelas 5.3 e 5.4, uma vez que as médias são estatisticamente diferentes. O desvio padrão também é apresentado nas tabelas para os casos de testes efetivos Normal e Dual para todos os programas.

Médias com letras distintas diferem-se pelo Teste de Tukey e pelo Teste de Duncan (p<0,05).

Tabela 5.3: Resumo das estatísticas (médias e desvios padrões) dos resultados da avaliação do Critério de Teste Mutação e Mutação Dual.

Programa	Casos de testes Efetivos				Equivalentes	
	Normal		Dual		Normal	Dual
CALCDATE	75,33[a]	± 6	37,53[b]	±2	714[a]	277[b]
CALEN	30,13[a]	±21	15[b]	±1	485[a]	10[b]
CALEND	26,53[a]	±1	23,93[b]	±3	541[a]	386[b]
CAL	76,93[a]	±23	22,96[b]	±5	326[a]	172[b]
JDAY	34,63[a]	±11	10,03[b]	±1	70[a]	34[b]
DATEPLUS	32,3[a]	±60	3[b]	±0	396[a]	391[b]

Médias com letras distintas diferem-se pelo Teste de Tukey e pelo Teste de Duncan (p<0,05).

Tabela 5.4: Resumo das estatísticas (médias e desvios padrões) Escore de Mutação do Cruzamento do resultado dos experimentos entre os dois critérios

Programa	Casos de testes efetivos				Escore de cruzamento			
	Normal		Dual		Normal		Dual	
CALCDATE	32,3[a]	±11	61,13[b]	±7	0,817401[a]	±0,013	0,652488[b]	±0,081
CALEN	12,7[a]	±1	14,23[b]	±4	0,892673[a]	±0,004	0,519887[b]	±0,023
CALEND	22,4[a]	±7	25,06[b]	±2	0,884740[a]	±0,023	0,518610[b]	±0,090
CAL	14,53[a]	±2	43,8[b]	±11	0,942834[a]	±0,019	0,445192[b]	±0,015
JDAY	9,23[a]	±1	24,9[b]	±5	0,959063[a]	±0,003	0,524340[b]	±0,042
DATEPLUS	3[a]	±0	8,56[b]	±2	0,784314[a]	±0,000	0,556609[b]	±0,024

5.6 Considerações finais

Neste capítulo foi apresentada a avaliação empírica do trabalho em questão e foram demonstrados os passos seguidos para a realização da pesquisa. Enfatizaram-se os resultados que apontaram o critério de Teste de Mutação Dual como o mais favorável quando comparado com o Critério de Mutação. Esses resultados foram fundamentados teoricamente nos fatores básicos necessários para comparar a adequação de um critério de teste, que são **custo, eficácia e dificuldade de satisfação**. Para os três fatores o Critério de Mutação Dual apresentou uma adequação melhor do que a Mutação Normal.

6 CONCLUSÕES

Atualmente, sistemas computadorizados estão presentes em várias áreas do conhecimento. Na engenharia de software, a atividade de teste tem como principal objetivo aumentar a confiabilidade e garantir a qualidade do software para que este seja apresentado com o mínimo de erros possíveis.

A atividade de teste tem sido apontada como uma etapa dentre as mais onerosas no desenvolvimento de software. É utilizada na tentativa de reduzir os custos associados ao teste, com a aplicação de técnicas e critérios que dão indicações de como testar o software com eficácia.

Com a aplicação de critérios de teste, torna-se necessário quantificar a atividade de teste, utilizando a análise de cobertura que indica quanto dos requisitos de um critério foi satisfeito em um teste (FABBRI, 1996). Essa avaliação quantitativa é dada pelo escore de mutação, que corresponde à razão entre o número de mutantes mortos pelo número de mutantes não-equivalentes gerados.

Neste trabalho foi feita uma análise empírica de dois critérios de teste, implementados na ferramenta *PROTEUM/IM* 2.0. A partir dessa análise foi desenvolvida uma metodologia para a determinação da avaliação experimental dos critérios de teste. Também foi feita uma análise estatística dos dados, por meio de análise de variância para testar as hipóteses relacionadas aos critérios de testes utilizados no estudo.

Algumas conclusões podem ser obtidas, a partir dos resultados encontrados.

Há diferença significativa em nível de significância de 5% entre os valores dos critérios dos testes Normal e Dual, para os seis diferentes programas testados.

Para o Critério de Teste Dual, o número de casos de testes requeridos foi inferior para os seis programas avaliados, implicando em menor **custo** para esse critério.

Para o Critério de Teste Dual, o número de mutantes mortos para os seis programas avaliados foi superior, com a aplicação de um número de casos de testes menor, demonstrando uma maior **eficácia**.

O resultado do cruzamento entre os dois critérios apresentou melhor resultado do escore de mutação do cruzamento para o critério Normal, em virtude da utilização do conjunto de casos de testes do critério Dual. Esses resultados enaltecem o Critério de Mutação Dual, com relação à adequação **dificuldade de satisfação**.

A repetição das sessões torna os dados mais confiáveis, o que foi comprovado com os resultados obtidos no trabalho, nos quais pode ser observada a mesma tendência no resultado para todos os fatores avaliados nos dois critérios, apontando uma adequação quanto ao **custo**, à **eficácia** e à **dificuldade de satisfação** favorável ao Critério Dual.

Contribuições deste Trabalho

Algumas contribuições deste trabalho devem ser destacadas, tais como:

- **Avaliação empírica dos critérios Análise de Mutação e Análise de Mutação Dual, avaliação dos conjuntos de casos de testes efetivos, avaliação dos mutantes equivalentes e avaliação do escore de cobertura.**
- **Determinação do conjunto de operadores essenciais de mutação (Apêndice G).**

Trabalhos Futuros

Os resultados obtidos motivam a continuidade dessa linha de pesquisa. Algumas possibilidades são:

- Conduzir outros experimentos que utilizem outros conjuntos de programas visando à comparação entre critérios de teste diferentes.

- Avaliar o **custo,** a **eficácia** e a **dificuldade de satisfação** com a utilização de outros operadores de mutação.
- Com base nas informações acumuladas, desenvolver heurísticas que contribuam para o trabalho realizado.

REFERÊNCIAS BIBLIOGRÁFICAS

BABBIE, Survey Research Methods, Wadsworth, ISBN 0-524-12672-3, 1990.

BARBOSA, E.;VINCENZI, A.; MALDONADO, J.C. Uma contribuição para Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C. Simpósio Brasileiro de Engenharia de Software – SBES'98, *Anais*. Maringá, Outubro 1998. p. 33-34.

BASILI, R. Quantitative Evaluation of Software Engineering Methodology. Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia, *Proceedings*, Vol. 1, 1985. p. 379-398.

_____. Software Development: A Paradigm for the future. Proceedings of the 13th Annual International Computer Software & Applications Conference. *Proceedings* COMPSAC'89, Orlando, Florida, USA, 1989. p. 471-485.

_____; REITER, R.Jr. A Controlled Experiment Quantitatively Comparing Software Development Approaches. *IEEE Transactions on Software Enginnering*. n.5, vol 7, 1981. p. 299-320.

BUDD, T. A. *Mutation Analysis: Ideas, Example, Problems and Prospects*. North-Holand Publishing Company, 1981.

CHAIM, M. L.; MALDONADO, J.C.; JINO, M. Processo de depuração depois do teste: definição e análise. Campinas: Embrapa Informática Agropecuária, 2002. 47 p.: il. — (*Boletim de Pesquisa e Desenvolvimento* / Embrapa Informática Agropecuária)

CONRADI,R. *et al*. *A Pragmatic Document Standards for an Experience Library: Roles, Documents, Contents and Structure*, Technical Report CS-TR-4235, University of Maryland, April 2001.

DELAMARO, M.E. *Proteum – Um Ambiente de Teste Baseado na Análise de Mutantes*. Dissertação de Mestrado. ICMC/USP, São Carlos, SP, Outubro 1993. p.113.

_____. *Mutação de Interface*: Um critério de Adequação Interprocedimental para o Teste de Integração. Tese de Doutorado, IFSC/USP, São Carlos, SP, Junho, 1997.

_____. *Interface Mutation*: An Approach for Integration Testing. *IEEE Transactions on Software Engineering,* v. 27, n. 3, 2001. p. 228-247.

_____. *Dual Mutation*: The "Save the mutants" approach. *Dual Mutation*: The "Save the mutants" approach. In: Simpósio Brasileiro de Qualidade de Software, 2004, Brasília. Simpósio Brasileiro de Qualidade de Software, 2004. p. 301-312.

_____ *et al*. *Interface Mutation Test Adequacy Criterion*: An Empirical Evaluation, 1998.

_____; MALDONADO, J.C. Uma Visão sobre a Aplicação da Análise de Mutantes. *Notas do ICMC*, n. 133, São Carlos, SP, Março 1993.

_____; MALDONADO, J.C.; MATHUR, A.P. Interface Mutation: An Approach for Integration Testing, *IEEE Transactions of Software Engeneering*, 1998. p 177-189.

_____; MALDONADO, .C.; VINCENZI, A.M.R. *Proteum/IM 2.0*: An Integrated Mutation Testing Environment, Mutation 2000 Symposium.**Anais**. San Jose, CA, 2000. p.124-134.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F.G. Hints on Test Data Selection: help for the practicing programmer. *Computer*, v. 11, n. 4, 1978. p. 34-41.

DEMILLO, R.A; OFFUTT, A.J. Constraint Based Test Data Generation. *IEEE Transactions on Software Engeenering*, vol 17, n.9, setembro 1991. p. 900-910

DEMMING, E. *Out of the Crisis*. MIT Centre for Advanced Engineering Study, MIT Press: Cambridge, Massachusetts, 1986.

DÓRIA, E.S. *Replicação de Estudos Empíricos em Engenharia de Software*. Dissertação de Mestrado. ICMC/USP, São Carlos- SP, 2001. p.154.

FABBRI, S.C.P.F. *A análise de mutantes no contexto de sistemas reativos*: uma contribuição para o estabelecimento de estratégias de teste e validação. Tese de doutorado, IFSC-USP, outubro 1996. p.237.

FRANKL,F. G. *The Use of Data Flow Information for the Selection and Evaluation of Software Test Data*. PhD thesis, Universidade de New York, New York, NY, October 1987.

GOEL, A. L. Software Reability Models: Assumptions, Limitations, and Aplicability. *IEEE Transaction on Software Engineering*, v.11, n.12, Dezembro, 1985.

HOWDEN, W. E. *Functional Program Testing and Analysis*. McGrall-Hill: New York, 1987.

JORGE, R.F. *et al*. *Relatório dos operadores de mutação implementados nas ferramentas Proteum* e *Proteum/IM*. Departamento de Ciências da Computação e Estatística – Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, 2001. p. 179. Relatório Técnico.

JORGE, R.F; DELAMARO, M.E. *Teste de Mutação: Subsídios para Redução do Custo de Aplicação*, In: VI Simpósio de Teses e Dissertações do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional do ICMC/USP, 2001, São Carlos. VI Simpósio de Teses e Dissertações do Programa de Pós-Graduação do ICMC/USP, 2001. p. 35-36.

MALDONADO, J.C. *Critérios Potenciais Usos*: uma contribuição ao Teste Estrutural de Software. Tese de Doutorado. DCA/FEE/UNICAMP,Campinas, SP, Julho, 1991. p 261,

MALDONADO, J.C. *et al*. Aspectos Teóricos e Empíricos de Teste de Cobertura de Software. In: SBC. (Org.). VI Escola de Informática da SBC - Região Sul. Curitiba, 1998, v. 1, p. -..

MATHUR, A.P.; WONG, W.E. Evaluation of the cost of alternative mutation strategies. In VII Simpósio Brasileiro de Engenharia de Software. *Anais*, Rio de Janeiro, RJ, Brazil, Out 1993. p. 320-335.

_____. An empirical comparison of data flow and mutation based test adequacy criteria. *The Journal of Software Testing, Verification, and Reliability*. Vol. 4, n.1, Março 1994. p.9-31.

MONTGOMERY, D.C. *Design and Analysis of Experiments*. 4. ed.John Wiley & Sons, 1997.

MYERS,G.J. *The Art of Software Testing*. Wiley: New York, 1979.

NESI, C.N. *Métodos alternativos para realização de testes de hipóteses em delineamento experimental*. Dissertação de mestrado, ESALQ/USP, Piracicaba, 2002. p.134.

NTAFOS, S. C. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*, vol.14, n.6, July 1988. p.868-873.

OFFUTT, A. J.; TEWARY K. P.; ZHANG, T. An experimental evaluation of data flow and mutation testing. *Software Practice and Experience*, vol.26, n.2, February 1996. p.165-176.

OFFUTT, A.J. *et al*. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering Methodology*. Vol.5, n.2, April 1996. p.99-118.

OGLIARI, P.J; ANDRADE, D.F. *Estatística básica para as ciências agronômicas e biológicas com noções de experimentação*. UFSC/CT/DIE, Florianópolis, 2005. 358p.

PRESSMAN, R.S. *Engenharia de Software*. 5. ed. McGraw-Hill: Rio Janeiro, 2002.

PFLEEGER, S. Experimental Design and Analysis in Software Engineering Part 1-5, ACM Sigsoft, *Software Engineering Notes*, Vol. 19, No. 4, pp. 16-20; Vol. 20, No.1, pp. 22-26; Vol. 20, No. 2, pp. 14-16; Vol. 20, No. 3, pp. 13-15; and Vol. 20, No. 4, pp. 14-17, 1994-1995.

PIMENTEL GOMES, F. *Curso de Estatística Experimental*. 13.ed. Piracicaba: Nobel, 1990. 467p.

RAPPS, S.; WEYUKER, E.J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, vol.11, n.4, April 1985. p. 367-375.

ROBSON, C. *Real World Research: A Resource for Social Scientists and Practitioners-Researchers*. Blackwell, 1993.

SIEGEL,S.; CASTELLAN,J.; *Nonparametric Statistics for the Behavioral Sciences*. 2. ed. McGraw-Hill International Editions, 1988.

SOUZA, S. R. S. *Avaliação do custo e eficácia do critério análise de mutantes na atividade de teste de software*. Dissertação de Mestrado, ICMC-USP, São Carlos -SP, Junho 1996.

SPOTTO, E.; MALDONADO, J.C.; JINO, M. Teste estrutural de software: uma abordagem para aplicações de banco de dados relacional. *Anais do XIV Simpósio Brasileiro de Engenharia de Software,* João Pessoa, Paraíba, 2000. v.I.

STALE, R.E. The Art of Case Study Research. Beverly Hills: SAGE Publications, 1995.

STATISTICAL ANALYSIS SYSTEMS INSTITUTE. *SAS/STAT user's guide*: version 6. 4.ed. Cary, NC: SAS Institute Inc., 1989. v. 2, 846p.

VERGÍLIO,S.R.; MALDONADO J. C.; JINO, M. Caminhos não-executáveis na automação das atividades de teste. *Anais*. In VI Simpósio Brasileiro de Engenharia de Software, Gramado, RS, Nov 1992. p.343-356.

_____. Uma estratégia para a geração de dados de teste. *Anais do VII Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, Outubro 1993. p. 307-319.

VINCENZI, A.M.R. *Subsídio para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*. ICMC/USP, São Carlos- SP, 1998. Dissertação de Mestrado. ICMC/USP. p.138.

WEYUKER, E.J. *The cost of data _ow testing*: an empirical study. *IEEE Transactions on Software Engineering*. Vol.16, n.2, February 1990. p.121-128.

WOHLIN, C. *et al*. *Experimentation in software engineering - an introduction*. Assinippi Park: Kluwer Academic Publishers, 2002. p.204.

WONG, W. E.; MATHUR, A.P.; MALDONADO, J.C. Mutation versus all-uses: An empirical evaluation of cost, strength, and effectiveness. In International Conference on Software Quality and Productivity, *Anais*, Hong Kong, December 1994. Chapman and Hall. p. 258-265.

YIN, R.K. *Case Study Research Design and Methods***.** Sage Publications: Beverly Hills, 1994.

PROGRAMAS

Programa *Calcdate*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/calcdate.c> Acesso em 07/03/2005.

Programa *Calen*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/> Acesso em 07/03/2005.

Programa *Calend*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/> Acesso em 07/03/2005.

Programa *Cal*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/> Acesso em 07/03/2005.

Programa *Jday*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/jday-jdate.c> Acesso em 07/03/2005.

Programa *Dateplus*. Disponível em: <http://ftp.unicamp.br/pub/unix-c/calendars/> Acesso em 07/03/2005.

Apêndice A:

Descrição da funcionalidade, perfil operacional e linhas de código do Programa Calcdate

PROGRAMA CALCDATE

O programa calcula a diferença entre duas datas ou uma data e um número de dias dependendo da opção desejada.

Argumentos:

calcdate mmddyy -o x - calcula data x dias depois da data original

calcdate mmddyy -d mmddyy – calcula número de dias entre as duas datas

Entradas válidas:

1o. parâmetro: 010100 ate 123199

2o. parâmetro: idem para data -36524 ate 36524 para offset

(obs: intervalo de valores válidos para opção –o)

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo**:**

95% dos Casos de Testes Requeridoss requeridosválidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:

50% com –d

50% com –o

5% dos casos não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

1% com um único parâmetro

1% sem parâmetro

14% com mês não válido 1.° parâmetro

14% com ano não válido 1.° parâmetro

14% com dia não válido 1.° parâmetro

14% com mês não válido 2.° parâmetro

14% com ano não válido 2.° parâmetro

14% com dia não válido 2.° parâmetro

14% com offset não válido (obs: intervalo de valores válidos para opção –o)

LINHAS DE CÓDIGO

Program: calcdate

Synopsis: calcdate mmddyy { -o [-]offset | -d mmddyy }

Purpose: Calculate the target date when given an initial date and an offset in days, or the difference between two dates when given two dates.

Options: There are two options as listed below. One or the other must be entered.

-o [-]offset Allows the user to enter an offset (in days) and returns the initial date plus the offset.

Negative offsets are permitted.

-d mmddyy Allows the user to enter a second date, and returns the difference between the two dates relative to the first date.

Author: Gordon A. Runkle ORI/Calculon
...uunet!men2a!1solaria!gordon

**********/

```
#include <stdio.h>

#define REG0
#define LEAP        1
#define NEG0
#define POS 1

/* Version and usage information.  usage is used to print */
char *version = "calcdate v2.01, 17 May 88, Gordon A. Runkle, ORI/Calculon\n";
char *usage =   "Usage: calcdate mmddyy { -o [-]offset | -d mmddyy }\n\n";
```

```
char *prog = "calcdate";



/* External variables for holding results of parsing of date values */
int month_1, day_1, year_1, month_2, day_2, year_2;

/* no_days[0] is for reg yrs, no_days[1] is for leap yrs */
int no_days[2][12] =
                {
                { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
                { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
                };



main(argc, argv)
int argc;
char *argv[];
{
extern char *version, *usage;


/* Test for the proper number of args. */
if (argc != 4)
        {
        fprintf(stderr, version);
        fprintf(stderr, usage);
        exit(-1);
        }

/* Tests first date for valid format and contents */
if (valid_date(argv[1], 1) != 0)
        exit(-1);                       /* The error message is in the function */

/* Test for an option flag in the second arg */
if (argv[2][0] != '-')
        {
        fprintf(stderr, usage);
        exit(-1);
        }

/* Make a decision based on the option */
if (argv[2][1] == 'o')
        {
        if (off_calc(argv[3]) != 0)
                exit(-1);
        }
else if (argv[2][1] == 'd')
        {
```

```
        if (valid_date(argv[3], 2) != 0)
                exit(-1);
        else
                {

                if (dates_calc() != 0)  /* no arg is needed - valid_date */
                        exit (-1);      /* takes care of the dates */
                }
        }
else
        {
        fprintf(stderr, usage);
        exit(-1);
        }

return(0);
}


/*************************************/
valid_date(tst_date, date_flag)
char tst_date[];
int date_flag;
{
extern int month_1, day_1, year_1;
extern int month_2, day_2, year_2;
extern char *prog;

int i, leap_flag, month, day, year;


if (strlen(tst_date) != 6)
        {
        fprintf(stderr, "%s: date must be entered as mmddyy\n", prog);
        return(-1);
        }

for (i = 0; i < 6; i++)
        if (tst_date[i] < '0' || tst_date[i] > '9')
                {
                fprintf(stderr, "%s: date must be entered as mmddyy\n", prog);
                return(-1);
                }


sscanf(tst_date, "%2d%2d%2d", &month, &day, &year);

if (month > 12)
        {
        fprintf(stderr, "%s: invalid month %d\n", prog, month);
```

```
        return(-1);
        }

if (year % 4 == 0)
        leap_flag = LEAP;

  else
        leap_flag = REG;

if (day > no_days[leap_flag][month - 1])
        {
        fprintf(stderr, "%s: invalid day %d\n", prog, day);
        return(-1);
        }

/* This determines where our carefully-checked values are stored */
if (date_flag == 1)
        {
        day_1 = day;
        month_1 = month;
        year_1 = year;
        }
else if (date_flag == 2)
        {
        day_2 = day;
        month_2 = month;
        year_2 = year;
        }
else
        {
        fprintf(stderr, "%s: unexpected error assigning date values\n", prog);
        return(-1);
        }

return(0);
}


/*************************************/
off_calc(offset)
char offset[];
{
extern int no_days[2][12];
extern int month_1, day_1, year_1;
extern char *prog;

int atoi();
int i_offset, n_month, n_day, n_year, month_bal;
int i, leap_flag;
```

```
char newdate[7];


/* This checks for a valid offset value.  Negative values are allowed
  and checked for.  It stops at the first null. */
for (i = 0; i < 4; i++)

        {
        if (offset[i] == '\0')
                break;

        if (i == 0 && offset[i] == '-')
                continue;

        if (offset[i] < '0' || offset[i] > '9')
                {
                fprintf(stderr, "%s: offset must be entered as an integer\n", prog);
                exit(-1);
                }
        }

i_offset = atoi(offset);

/* This is the beginning of the neat stuff.  I hope it works! */
/* leap year is when =>>  year % 4 == 0  */

n_year = year_1;      /* the *_1 is used, as this is the value of the */
n_month = month_1;          /* first date entered */
n_day = day_1;

if (i_offset >= 0)
        {
        while (i_offset > 0)
                {
                if (n_year % 4 == 0)
                        leap_flag = LEAP;
                  else
                        leap_flag = REG;

                month_bal = no_days[leap_flag][n_month - 1] - n_day;

                if (i_offset > month_bal)
                        {
                        i_offset -= month_bal;
                        n_month++;

                        if (n_month > 12)
                                {
                                n_month = 1;
                                n_year++;
```

```
                        if (n_year > 99)
                                n_year = 0;
                        }

                n_day = 0;
                }

         else
                {
                n_day += i_offset;
                i_offset = 0;
                }
        }
    }
else
    {
    while (i_offset < 0)        /* this loop processes neg offsets */
        {

        if (n_year % 4 == 0)
                leap_flag = LEAP;
         else
                leap_flag = REG;

        month_bal = n_day - 1;

        if (abs(i_offset) > month_bal)
                {
                i_offset += month_bal;
                n_month--;

                if (n_month < 1)
                        {
                        n_month = 12;
                        n_year--;

                        if (n_year < 0)
                                n_year = 99;
                        }

                n_day = no_days[leap_flag][n_month - 1] + 1;
                }
         else
                {
                n_day += i_offset;
                i_offset = 0;
                }
        }
    }
```

```
sprintf(newdate, "%2d%2d%2d", n_month, n_day, n_year);

for (i = 0; i < 7; i++)
        if (newdate[i] == ' ')
                newdate[i] = '0';

fprintf(stdout, "%s\n", newdate);

return(0);
}


/*************************************/
dates_calc()
{
extern int no_days[2][12];
extern int month_1, day_1, year_1, month_2, day_2, year_2;


int first_rec = 0, curr_offset = 0;
int leap_flag, sign_flag;
int start_day, start_month, start_year, end_day, end_month, end_year;


/****
        This section determines which date is later, so that the program
        may evaluate the earlier one first.  There is a flag set to indicate
        what sign the end result should have based on whether the first date
        entered is earlier or later than the second.
****/

/* set the default sign */
sign_flag = NEG;

if (year_1 < year_2)
        sign_flag = POS;
  else
        if (year_1 == year_2 && month_1 < month_2)
                sign_flag = POS;
          else
                if (year_1 == year_2 && month_1 == month_2 && day_1 < day_2)
                        sign_flag = POS;

/* This makes the earlier date be set to start_* */
if (sign_flag == POS)
        {
        start_day = day_1;
        start_month = month_1;
        start_year = year_1;
```

```
        end_day = day_2;
        end_month = month_2;
        end_year = year_2;
        }
 else
        {
        start_day = day_2;
        start_month = month_2;
        start_year = year_2;

        end_day = day_1;
        end_month = month_1;
        end_year = year_1;
        }

/* The calculations below keep incrementing curr_offset and start_* until
  start_* == end_*  */

for (;;)
        {
        if (start_year % 4 == 0)
                leap_flag = LEAP;
         else
                leap_flag = REG;

        if (first_rec == 0)
                {
                /* This is for when the month and year start out the same, and
                   the user just wants the days (ie.  051688 052688 */
                if (start_month == end_month && start_year == end_year)
                        {
                        curr_offset = end_day - start_day;
                        break;
                        }

                curr_offset = no_days[leap_flag][start_month - 1] - start_day;
                first_rec = 1;
                }
         else if (start_month == end_month && start_year == end_year)
                {
                curr_offset += end_day;
                break;                  /* This is the end of it */
                }
         else
                curr_offset += no_days[leap_flag][start_month - 1];


        start_month++;

        if (start_month > 12)
```

```
                {
                start_month = 1;
                start_year++;

                if (start_year > 99)
                        start_year = 0;
                }
        }



if (sign_flag == NEG)
        curr_offset = -curr_offset;

fprintf(stdout, "%d\n", curr_offset);

return(0);
}
/* end of calcdate.c  -=gordon=- */
```

Apêndice B:

Descrição da funcionalidade, perfil operacional e linhas de código do Programa Calen

PROGRAMA CALEN

Este programa gera calendários do tamanho da tela, embora o objetivo é gerar esses calendários para impressoras matriciais. O programa aceita três parâmetros de entradas mais alguns argumentos.

Argumentos:

calen y – mostra calendário do ano y

calen m y – mostra calendário do mês m, ano y

calen m y n – mostra n meses a partir do mês m, ano y

Outros argumentos aceitos:

-l -r : justificação à esquerda e direita

-u -m : só maiúsculas ou misturadas

-o[seq] : usa "seq" para destacar texto

-bN : adiciona N linhas em branco após fim de página

Entradas válidas:

- Disposição das opções de entrada e parâmetros :

As opções podem ser:

-l (justifica as datas a esquerda) ou

-r (justifica as datas a direita) ou

nenhuma dessas opções mais

-m (deixa os dias da semana em maiúsculas e minúsculas) ou

-u (deixa os dias da semana em maiúsculas) ou

nenhuma dessas opções mais

-o[seq_caract] (seqüência de caracteres que forma o nome do mes) mais

-bN (acrescenta linhas em branco entre os calendários) mais

nenhuma dessas opções.

Os parâmetros podem ser

1.° parâmetro:

Pode ser somente o ano do calendário de 1753 a 9999 ou o mês de 1 a 12.

2.° parâmetro:

Se o primeiro parâmetro for um mês o segundo será um ano, também de 1753 a 9999

3.° parâmetro:

Somente pode ser usado se usamos o formato mês e ano , esse terceiro parâmetro são o numero de meses que queremos gerar o calendário a partir do mês ano especificados nos 1.º e 2.º parâmetros.

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo**:**

95% dos Casos de Testes Requeridoss requeridossão válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:
34% com um parâmetro de ano
33% com um parâmetro de mês e um de ano
33% com um parâmetro de mês, um de ano e um de numero de meses

Obs: Para cada um dos casos válidos acima citados, cada um poderá ter a probabilidade de 20% de possuir uma das opções listadas abaixo:

20% de probabilidade de não possuir nenhuma opção.

20% de probabilidade de possuir somente uma opção.
20% de probabilidade de possuir duas opções.
20% de probabilidade de possuir três.
20% de probabilidade de possuir quatro opções.

5% dos casos são não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

50% dos casos não válidos tem a probabilidade de ser:

10% com um único parâmetro ano inválido
9% com o parâmetro mês inválido e o ano válido
9% com o parâmetro mês válido e o ano inválido
9% com os dois parâmetros mês e ano inválidos
9% com o parâmetro mês inválido e os parâmetros ano e número de meses válidos
9% com o parâmetro mês válido, o ano inválido e o numero de meses válido
9% com o parâmetro mês válido, o ano válido e o número de meses inválido
9% com o parâmetro mês inválido, o ano inválido e o número de meses válido
9% com o parâmetro mês inválido, o ano válido e número de meses inválido
9% com o parâmetro mês válido, o ano inválido e o número de meses inválido
9% com os parâmetros mês, ano e número de meses inválido

50% dos casos não válidos tem a probabilidade de ser alguma das opções abaixo somente se estiverem acompanhadas de uma opção invalida, isto é, os parâmetros podem estar corretos, no entanto o que ira caracterizá-lo como inválido e a opção que o acompanha, pois esta será invalida.

34% com um parâmetro de ano
33% com um parâmetro de mês e um de ano
33% com um parâmetro de mês, um de ano e um de numero de meses

LINHAS DE CÓDIGO

```
/*
 *      Calendar program - one month per page
 *
 *      Originally written in FORTRAN-IV for GE Timesharing, 10/65
 *      Re-coded in C for UNIX, 3/83
 *
 *      Author: AW Rogers
 *
 *      Parameters:
 *
 *              calen yy                generates calendar for year yy
 *
 *              calen mm yy [len]       generates calendar for len months
 *                                      (default = 1) starting with mm/yy
 *
 *      Option flags (must precede params):
 *
 *              -l                      left-justify dates (default)
 *              -r                      right-justify dates
 *              -m                      mixed-case output (default)
 *              -u                      upper-case output
 *              -o[seq]                 use "seq" as overstrike sequence
 *                                       for heading (default: HIX)
 *              -bN                     add N blank lines after form-feed
 *
 *              Output is to standard output.
 *
 */

#include <stdio.h>
#include <ctype.h>

#define FALSE   0
#define TRUE    1

#define JAN             1                       /* significant months/years */
#define FEB 2
#define DEC 12
#define MINYR       1753
#define MAXYR       9999

#define SOLID       0                       /* pseudo-enumeration for line styles */
#define OPEN        1
```

```
#define LEFT        0                   /* ... and justification of dates     */
#define RIGHT       1

#define MIXED       0                   /* ... and case of output text        */

#define UPPER       1

#define     OVERSTRIKE      "HIX"       /*  overstrike  sequence  for
month/year */
#define MAX_OVERSTR  3                  /* maximum overstrikes permitted */

#define isLeap(y) ((y) % 4 == 0 && ((y) % 100 != 0 || (y) % 400 == 0)) /*  leap  year
macro */

typedef struct                          /* info for a single month */
   {
   int mm;
   int          yy;
   char         mmname[10];
   char         dates[6][7][3];
   } monthRec;

typedef monthRec *mptr;                 /* pointer to above struct */

/* globals corresponding to command-line flags */

int just = LEFT;                    /* default justification of dates */
int ocase = MIXED;                  /* default case for output */
int nblank = 0;                         /* default blank lines after FF */
char *seq = OVERSTRIKE;                         /* default overstrike sequence */


/*
 *      Main - gets and validates parameters, opens output file, executes
 *      loop to fill and print months of calendar, closes output file
 */
main(argc, argv)
   int argc;
   char *argv[];
   {
   int nmonths;                     /* consecutive months to print */
   int badopt = FALSE;                  /* flag set if bad option */
   int badpar = FALSE;                  /* flag set if bad param  */
   monthRec mRec[3];                    /* space for main and small calendars */
   mptr prev = &mRec[0],            /* pointers to calendars (initially) */
```

```
    curr = &mRec[1],
    next = &mRec[2],
    temp;

/* Get command line flags */

while (argc > 1 && argv[1][0] == '-')
   {
   switch (argv[1][1])
       {
       case 'b':
            sscanf(&argv[1][2], "%d", &nblank);
            break;
       case 'l':
            just = LEFT;
            break;
       case 'r':
            just = RIGHT;
            break;
       case 'm':
            ocase = MIXED;
            break;
       case 'u':
            ocase = UPPER;
            break;
       case 'o':
            if (argv[1][2] != '\0')
               seq = &argv[1][2];
            break;
       default:
            fprintf(stderr, "Invalid flag: %s\n", argv[1]);
            badopt = TRUE;
            break;
       }
   argv++;
    argc--;
    }

if (badopt)
    fprintf(stderr, "Valid flags are -b -l -m -o -r -u\n");

/* Get and validate parameters */

if (argc == 2)                  /* only one arg - treat as yy */
    {
    sscanf(argv[1], "%d", &curr->yy);
    curr->mm = JAN;
    nmonths = 12;
    }
```

```
else if (argc >= 3)          /* two or more - treat as mm yy [len] */
    {
    sscanf(argv[1], "%d", &curr->mm);
    sscanf(argv[2], "%d", &curr->yy);
    if (argc >= 4)


       sscanf(argv[3], "%d", &nmonths);
    }

else                         /* none specified - get interactively */
    {
    fprintf(stderr, "Enter calendar specs (month year length): ");
    scanf("%d %d %d", &curr->mm, &curr->yy, &nmonths);
    }

if (curr->yy > 0 && curr->yy < 100)          /* nn -> 19nn */
    curr->yy += 1900;

if (nmonths < 1)                             /* default for month count */
    nmonths = 1;

if (curr->mm < JAN || curr->mm > DEC)        /* validate month/year */
  {
  fprintf(stderr, "Month %d not in range %d .. %d\n", curr->mm, JAN, DEC);
  badpar = TRUE;
  }

if (curr->yy < MINYR || curr->yy > MAXYR)
  {
  fprintf(stderr, "Year %d not in range %d .. %d\n", curr->yy, MINYR,
          MAXYR);
  badpar = TRUE;
  }

if (badpar)          /* quit if month or year invalid */
  exit(1);

/* fill in calendars for previous and current month */

prev->mm = (curr->mm == JAN) ? DEC : curr->mm - 1;
prev->yy = (curr->mm == JAN) ? curr->yy - 1 : curr->yy;
fillCalendar(prev);

fillCalendar(curr);

/*
 * Main loop: print each month of the calendar (with small calendars for
 * the preceding and following months in the upper corners).  The current
 * and next months' calendars can be reused the following month; only
```

```
         * the 'next' calendar need be recalculated each time.
         */

        for (; nmonths > 0 && curr->yy <= MAXYR; nmonths--)       /* main loop */
            {
            next->mm = (curr->mm == DEC) ? JAN : curr->mm + 1;
            next->yy = (curr->mm == DEC) ? curr->yy + 1 : curr->yy;

            fillCalendar(next);                     /* fill in following month */

            printCalendar(prev, curr, next);

            temp = prev;                            /* swap the three months */
            prev = curr;
            curr = next;
            next = temp;
            }

        }

    /*
     *      Print the calendar for the current month, generating small calendars
     *      for the previous and following months in the upper corners and the
     *      month/year (in large characters) centered at the top.
     */
    printCalendar(prev, curr, next)
        mptr prev;                  /* Previous month (upper-left corner)  */
        mptr curr;                  /* Current month (main calendar)       */
        mptr next;                  /* Next month (upper-right corner)     */
        {
        int nchars, i, j;
        static char *mc_wkday[] =
            {
            " Sunday  ", " Monday  ", " Tuesday ", "Wednesday", "Thursday ",
            " Friday  ", "Saturday "
            };
        static char *uc_wkday[] =
            {
            "  SUNDAY    ",  "  MONDAY    ",  "  TUESDAY  ",  "WEDNESDAY",
"THURSDAY ",
            " FRIDAY  ", "SATURDAY "
            };

        char **wkday;                               /* pointer to one of above */
        char *blanks = "                    ";   /* 21 blanks for centering */
        char *padding;                              /* Pointer into 'blanks'   */
        char monthAndYear[20];                      /* Work area               */
        char *ovr;                                  /* overstrike sequence             */

        nchars = strlen(curr->mmname);          /* set up month/year heading */
```

```
padding = blanks + (3 * (nchars - 3));    /* and center it              */
sprintf(monthAndYear, "%s%5d", curr->mmname, curr->yy);

printf("\f\n");                    /* print month/year in large chars */
for (i = 0; i < nblank; i++)
    printf("\n");

for (i = 0; i < 9; i++)                /* surrounded by small calendars   */

   {
    for (ovr = seq;                /* overstruck lines first */
        ovr < seq + MAX_OVERSTR - 1 && *(ovr+1);
        ovr++)
        {
        printf("%20s%s", " ", padding);
        printHdr(monthAndYear, i, *ovr);
        printf("\r");
        }
    printSmallCal(prev, i);                /* then small calendars, etc. */
    printf("%s", padding);
    printHdr(monthAndYear, i, *ovr);
    printf(" %s", padding);
   printSmallCal(next, i);
   printf("\n");
   }

printf("\n");                      /* print the weekday names */
print_line(1, SOLID);
print_line(1, OPEN);
printf("  ");
wkday = ocase == UPPER ? uc_wkday : mc_wkday;
for (j = 0; j < 7; j++)
   printf("|%13.9s    ", wkday[j]);
printf("|\n");
print_line(1, OPEN);

for (i = 0; i < 4; i++)                /* print first four rows */
   {
   print_line(1, SOLID);
   print_dates(curr, i, just);
   print_line(7, OPEN);
   }

print_line(1, SOLID);              /* print bottom row */
print_dates(curr, 4, just);
print_line(2, OPEN);
print_divider(curr->dates[5]);     /* divider for 23/30, 24/31 */

print_line(3, OPEN);
print_dates(curr, 5, !just);       /* print 6th line (30/31) at bottom */
```

```
    print_line(1, SOLID);
    }

/*
 *      Fill in the month name and date fields of a specified calendar record
 *      (assumes mm, yy fields already filled in)
 */
fillCalendar(month)
    mptr month;                         /* Pointer to month info record */

    {
    typedef struct              /* Local info about months */
        {
        char *name[2];                  /* Name of month (mixed/upper-case)        */
        int offset[2];          /* Offset of m/1 from 1/1 (non-leap/leap) */
        int length[2];          /* Length (non-leap/leap)                  */
        } monthInfo;

    static monthInfo info[12] = {
        { {"January",  "JANUARY"},          {0, 0}, {31, 31} },
        { {"February","FEBRUARY"},          {3, 3}, {28, 29} },
        { {"March",    "MARCH"},    {3, 4}, {31, 31} },
        { {"April",    "APRIL"},    {6, 0}, {30, 30} },
        { {"May",      "MAY"},              {1, 2}, {31, 31} },
        { {"June",     "JUNE"},     {4, 5}, {30, 30} },
        { {"July",     "JULY"},     {6, 0}, {31, 31} },
        { {"August",   "AUGUST"}, {2, 3}, {31, 31} },
        { {"September",        "SEPTEMBER"},       {5, 6}, {30, 30} },
        { {"October", "OCTOBER"},          {0, 1}, {31, 31} },
        { {"November",         "NOVEMBER"},        {3, 4}, {30, 30} },
        { {"December",         "DECEMBER"},        {5, 6}, {31, 31} }
    };

    int i, first, last, date = 0, y = month->yy, m = month->mm-1;
    int leap = isLeap(y);

    first = (y + (y-1)/4 - (y-1)/100 + (y-1)/400 + info[m].offset[leap]) % 7;
    last = first + info[m].length[leap] - 1;

    for (i = 0; i < 42; i++)                    /* fill in the dates */
        if (i < first || i > last)
            strcpy(month->dates[i/7][i%7], "  ");
        else
            sprintf(month->dates[i/7][i%7], "%2d", ++date);

    strcpy(month->mmname, info[m].name[ocase]);         /* copy name of month */
    }

/*
 *      Print one line of a small calendar (previous and next months in
```

```
 *      upper left and right corners of output)
 */
printSmallCal(month, line)
   mptr month;                          /* Month info record pointer  */
   int line;                    /* Line to print (see below)    */
   {
   int i;

   switch (line)
       {

       case 0:          /* month/year at top */
          printf("   %-10s%4d   ", month->mmname, month->yy);
          break;
       case 1:          /* blank line */
          printf("%20s", " ");
          break;
       case 2:          /* weekdays */
          printf(ocase == UPPER ? "SU MO TU WE TH FR SA" :
                                  "Su Mo Tu We Th Fr Sa");
          break;
       default:         /* line of calendar */
          for (i = 0; i <= 5; i++)
                printf("%s ", month->dates[line-3][i]);
          printf("%s", month->dates[line-3][6]);
          break;
       }
   }

/*
 *      Print n lines in selected style
 */
print_line(n, style)
   int n;                       /* Number of lines to print (> 0)   */
   int style;                   /* SOLID or OPEN                    */
   {
   int i;
   char *fmt1 = (style == SOLID) ? "+----------------" :
                                   "|                " ;
   char *fmt2 = (style == SOLID) ? "+\n" : "|\n" ;

   for (; n > 0; n--)
       {
       printf("  ");
       for (i = 0; i < 7; i++)
          printf(fmt1);
       printf(fmt2);
       }
   }
```

```
/*
 *      Print line of large calendar (open w/left- or right-justified dates)
 */
print_dates(month, line, just)
   mptr month;                          /* Month info record pointer  */
   int line;                    /* Line to print (0-5)              */
   int just;                    /* justification (LEFT / RIGHT)       */
   {
   int i;

   char *fmt = (just == LEFT) ? "| %-16s" : "|%16s " ;

   printf("  ");
   for (i = 0; i < 7; i++)
       printf(fmt, month->dates[line][i]);
   printf("|\n");
   }

/*
 *      Print divider between 23/30 and 24/31
 */
print_divider(dates)
   char dates[7][3];
   {
   int j;

   printf("  ");
   for (j = 0; j < 7; j++)
      if (strcmp(dates[j], "  ") == 0)
          printf("|                ");
      else
          printf("|________________");
   printf("|\n");
   }


/*
 *      Print LS 6 bits of n (0 = ' '; 1 = selected non-blank)
 */
decode(n, c)
   int n;                               /* Number to print (0-31) */
   char c;
   {
   int msk = 1 << 5;

   for (; msk; msk /= 2)
       printf("%c", (n & msk) ? c : ' ');
   }
```

```
/*
 *      Print one line of string in large characters
 */
printHdr(str, line, c)
   char *str;                    /* string to print                  */
   int line;                     /* line to print (0-8; else blanks) */
   char c;                       /* output character to use          */
   {

   /* 5x9 dot-matrix representations of A-Z, a-z, 0-9 */

   static char uppers[26][9] =
       {
       {14, 17, 17, 31, 17, 17, 17,  0,  0},   {30, 17, 17, 30, 17, 17, 30,  0,  0},   /*
AB */
       {14, 17, 16, 16, 16, 17, 14,  0,  0},   {30, 17, 17, 17, 17, 17, 30,  0,  0},   /*
CD */
       {31, 16, 16, 30, 16, 16, 31,  0,  0},   {31, 16, 16, 30, 16, 16, 16,  0,  0},   /* EF
*/
       {14, 17, 16, 23, 17, 17, 14,  0,  0},   {17, 17, 17, 31, 17, 17, 17,  0,  0},   /*
GH */
       {31,  4,  4,  4,  4,  4, 31,  0,  0},       { 1,  1,  1,  1,  1, 17, 14,  0,  0},       /*  IJ
*/
       {17, 18, 20, 24, 20, 18, 17,  0,  0},   {16, 16, 16, 16, 16, 16, 31,  0,  0},   /* KL
*/
       {17, 27, 21, 21, 17, 17, 17,  0,  0},   {17, 17, 25, 21, 19, 17, 17,  0,  0},   /*
MN */
       {14, 17, 17, 17, 17, 17, 14,  0,  0},   {30, 17, 17, 30, 16, 16, 16,  0,  0},   /* OP
*/
       {14, 17, 17, 17, 21, 18, 13,  0,  0},   {30, 17, 17, 30, 20, 18, 17,  0,  0},   /*
QR */
       {14, 17, 16, 14,  1, 17, 14,  0,  0},   {31,  4,  4,  4,  4,  4,  4,  0,  0},       /* ST
*/
       {17, 17, 17, 17, 17, 17, 14,  0,  0},   {17, 17, 17, 17, 17, 10,  4,  0,  0},   /*
UV */
       {17, 17, 17, 21, 21, 21, 10,  0,  0},   {17, 17, 10,  4, 10, 17, 17,  0,  0},   /*
WX */
       {17, 17, 17, 14,  4,  4,  4,  0,  0},       {31,  1,  2,  4,  8, 16, 31,  0,  0}       /* YZ
*/
       };

   static char lowers[26][9] =
       {
       { 0,  0, 14,  1, 15, 17, 15,  0,  0},       {16, 16, 30, 17, 17, 17, 30,  0,  0},   /*  ab
*/
       { 0,  0, 15, 16, 16, 16, 15,  0,  0},       { 1,  1, 15, 17, 17, 17, 15,  0,  0},       /*  cd
*/
       { 0,  0, 14, 17, 31, 16, 14,  0,  0},       { 6,  9, 28,  8,  8,  8,  8,  0,  0},       /*  ef
```

```
*/
        { 0,  0, 14, 17, 17, 17, 15,  1, 14},    {16, 16, 30, 17, 17, 17, 17,  0,  0},   /*  gh
*/
        { 4,  0, 12,  4,  4,  4, 31,  0,  0},    { 1,  0,  3,  1,  1,  1,  1, 17, 14},   /*   ij
*/
        {16, 16, 17, 18, 28, 18, 17,  0,  0},    {12,  4,  4,  4,  4,  4, 31,  0,  0},   /*   kl
*/
        { 0,  0, 30, 21, 21, 21, 21,  0,  0},    { 0,  0, 30, 17, 17, 17, 17,  0,  0},   /* mn
*/
        { 0,  0, 14, 17, 17, 17, 14,  0,  0},    { 0,  0, 30, 17, 17, 17, 30, 16, 16},   /*  op
*/
        { 0,  0, 15, 17, 17, 17, 15,  1,  1},    { 0,  0, 30, 17, 16, 16, 16,  0,  0},   /* qr

    */
        { 0,  0, 15, 16, 14,  1, 30,  0,  0},    { 8,  8, 30,  8,  8,  9,  6,  0,  0},   /*   st
*/
        { 0,  0, 17, 17, 17, 17, 14,  0,  0},    { 0,  0, 17, 17, 17, 10,  4,  0,  0},   /*  uv
*/
        { 0,  0, 17, 21, 21, 21, 10,  0,  0},    { 0,  0, 17, 10,  4, 10, 17,  0,  0},   /* wx
*/
        { 0,  0, 17, 17, 17, 17, 15,  1, 14},    { 0,  0, 31,  2,  4,  8, 31,  0,  0}    /*  yz
*/
        };

    static char digits[10][9] =
        {
        {14, 17, 17, 17, 17, 17, 14,  0,  0},    { 2,  6, 10,  2,  2,  2, 31,  0,  0},   /*  01
*/
        {14, 17,  2,  4,  8, 16, 31,  0,  0},    {14, 17,  1, 14,  1, 17, 14,  0,  0},   /*  23
*/
        { 2,  6, 10, 31,  2,  2,  2,  0,  0},    {31, 16, 16, 30,  1, 17, 14,  0,  0},   /*  45
*/
        {14, 17, 16, 30, 17, 17, 14,  0,  0},    {31,  1,  2,  4,  8, 16, 16,  0,  0},   /*  67
*/
        {14, 17, 17, 14, 17, 17, 14,  0,  0},    {14, 17, 17, 15,  1, 17, 14,  0,  0}    /*  89
*/
        };

    char ch;

    for ( ; *str; str++)
        {
        ch = (line >= 0 && line <= 8) ? *str : ' ';
        if (isupper(ch))
            decode(uppers[ch-'A'][line], c);
        else if (islower(ch))
            decode(lowers[ch-'a'][line], c);
        else if (isdigit(ch))
            decode(digits[ch-'0'][line], c);
```

```
    else
        decode(0, c);
    }
}
```

Apêndice C:

Descrição da funcionalidade, perfil operacional e linhas de código do programa Calend

PROGRAMA CALEND

Este programa apresenta calendários para serem visualizados na tela. O programa aceita dois parâmetros:

mês que será gerado o calendário

ano da data que será gerado o calendário.

Argumentos:

calend mm – calendário do mês mm, ano corrente

calend yyyy – calendário do ano yyyy

calend mm yyyy – calendário do mês mm, ano yyyy

calend yyyy mm – calendário do mês mm, ano yyyy

Entradas Válidas:

1º Parâmetro:

Se forem valores de 1 a 12 -

Então é mostrado o calendário do mês indicado mais o anterior e o posterior do ano corrente

Senão, se for passado valores entre 13 e 9999

Então o valor passado é assumido como um ano e é mostrado o calendário daquele ano

Senão

Como nenhum valor foi passado o programa considera que o mês e ano a ser utilizado são os correntes

FimSe

FimSe

2º Parâmetro:

O Segundo Parâmetro necessariamente será um ano se o primeiro for um valor de 1 a 12, esse parâmetro poderá estar no intervalo de 0 a 9999. Agora se quisermos que o segundo parâmetro corresponda ao mês teremos que usar os valores para ano que estão no conjunto que corresponde entre 13 a 9999.

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo:

95% dos Casos de Testes Requeridoss requeridossão válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa.

Dos casos válidos:

4% com nenhum parâmetro

24% somente com valores de 1 a 12 no primeiro parâmetro

24% somente com valores de 13 a 9999 no primeiro parâmetro

24% com valores de 1 a 12 no primeiro parâmetro e de 0 a 9999 no segundo parâmetro

24% com de 13 a 9999 no primeiro parâmetro e 1 a 12 no segundo parâmetro

5% dos casos são não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

24% somente com um valor que não esteja ente 1 a 9999

24% com um valor válido de mês no primeiro parâmetro e um parâmetro inválido para ano significa que o ano não pode estar entre 0 e 9999

24% com um valor válido para ano no primeiro parâmetro, significa que o parâmetro deve ser 13 a 9999, e um parâmetro inválido de mês

28% com os dois parâmetros inválidos, sendo o primeiro corresponde ao mês (significando que os valores não podem ser de 1 a 12, mais também não podem estar no conjunto de parâmetros válidos do ano

LINHAS DE CÓDIGO

```
/*
 *                      C A L E N D A R
 *
 * Usage:
 *      calend MM               If small, it's a month, if large, a year.
 * or
 *      calend YYYY MM              year/month
 * or
 *      calend MM YYYY
 */

/*)BUILD
*/

#ifdef DOCUMENTATION

title   calend Print a Calendar
index calend Print a Calendar

usage

        calend [year] [month]

description

        When invoked without arguments, calend prints a
        calendar for the preceeding, current, and next
        months of the current year.

        If a month is given (a value from 1 through 12),
        it prints the three months centered on the requested
        month.  For example,

           calend 12

        Prints November and December for this year, and
        January for next year.

        If a year is given, it prints a calendar for the
        entire year:

           calend 1985

        If both values are given, it prints the three months
        centered on the indicated date:
```

```
   calend 1752 9
   calend 9 1752
```

Note that a three or four digit number will always be taken as a year. A one or two digit number will be either a month or a year in the 20th century:

```
   calend 78          (equals calend 1978)
   calend 0078        (early common era)
```

bugs

The change from the Julian to Gregorian calendars follows usage in England and her colonies. Options should be provided to process the change for other countries (and localities). This is, however, a fairly complex task with little payback.

The year didn't always start in January.

references

Enclycopaedia Brittanica, 13th edition, Vol. 4, p. 987 et. seq.

author

Martin Minow

```
#endif

#include <stdio.h>
#include <time.h>
#ifdef      decus
int     $$narg = 1;                 /* Don't prompt                 */
#endif
#ifdef vms
#include            ssdef
#define     IO_ERROR    SS$_ABORT
#define     IO_SUCCESSSS$_NORMAL
#endif
#ifndef     IO_ERROR
#define     IO_SUCCESS0             /* Unix definitions             */
#define     IO_ERROR    1
#endif
#define     EOS     0

#define     ENTRY_SIZE      3           /* 3 bytes per value        */
#define DAYS_PER_WEEK       7           /* Sunday, etc.
```

```
*/
  #define       WEEKS_PER_MONTH     6               /* Max. weeks in a month
*/

  #define       MONTHS_PER_LINE     3               /* Three months across
        */
  #define       MONTH_SPACE     3               /* Between each month
*/

  /*
   * calendar() stuffs data into layout[],
   * output() copies from layout[] to outline[], (then trims blanks).
   */
  char
layout[MONTHS_PER_LINE][WEEKS_PER_MONTH][DAYS_PER_WEEK][ENT
RY_SIZE];
  char  outline[(MONTHS_PER_LINE * DAYS_PER_WEEK * ENTRY_SIZE)
          + (MONTHS_PER_LINE * MONTH_SPACE)
          + 1];

  char  *weekday = " S  M Tu  W Th  F  S";
  char  *monthname[] = {
        "???",                                  /* No month 0 */
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
  };

  main(argc, argv)
  int           argc;
  char          *argv[];
  {
        register int    month;
        register int    year;

        register int    arg1val;
        int             arg1len;
        int             arg2val;
        int             tvec[2];
        struct tm       *tm;

        time(&tvec[0]);
        tm = localtime(&tvec[0]);
        year = tm->tm_year + 1900;
        month = tm->tm_mon + 1;
        if (argc <= 1) {
          /*
           * No arguments mean do last, this, and next month
           */
          do3months(year, month);
        }
```

```
        else {
            arg1val = atoi(argv[1]);
            arg1len = strlen(argv[1]);
            if (argc == 2) {

                /*
                 * Only one argument, if small, it's a month.  If
                 * large, it's a year.  Note:
                 *      calend 0082    Year '82
                 *      calend 82      Year 1982
                 */
                if (arg1len <= 2 && arg1val <= 12)
                    do3months(year, arg1val);
                else {
                    if (arg1len <= 2 && arg1val > 0 && arg1val <= 99)
                        arg1val += 1900;
                    doyear(arg1val);
                }
            }
            else {
                /*
                 * Two arguments, allow 1980 12 or 12 1980
                 */
                arg2val = atoi(argv[2]);
                if (arg1len > 2)
                    do3months(arg1val, arg2val);
                else
                    do3months(arg2val, arg1val);
            }
        }
        exit(IO_SUCCESS);
}

doyear(year)
int             year;
/*
 * Print the calendar for an entire year.
 */
{
        register int      month;

        if (year < 1 || year > 9999)
            usage("year", year);
        printf("\n\n\n%35d\n\n", year);
        for (month = 1; month <= 12; month += MONTHS_PER_LINE) {
            printf("%12s%23s%23s\n",
                    monthname[month],
                    monthname[month+1],
                    monthname[month+2]);
            printf("%s  %s  %s\n", weekday, weekday, weekday);
```

```
            calendar(year, month+0, 0);
            calendar(year, month+1, 1);
            calendar(year, month+2, 2);
            output(3);
#if MONTHS_PER_LINE != 3

            << error, the above won't work >>
#endif
        }
        printf("\n\n\n");
}

domonth(year, month)
int             year;
int             month;
/*
 * Do one specific month -- note: no longer used
 */
{
        if (year < 1 || year > 9999)
            usage("year", year);
        if (month <= 0 || month > 12)
            usage("month", month);
        printf("%9s%5d\n\n%s\n", monthname[month], year, weekday);
        calendar(year, month, 0);
        output(1);
        printf("\n\n");
}

do3months(thisyear, thismonth)
int             thisyear;
register int    thismonth;
/*
 * Do last month, this month, and next month.  The parameters
 * are guaranteed accurate. (and year will not be less than 2 nor
 * greater than 9998).
 */
{
        int             lastmonth;
        int             lastyear;
        int             nextmonth;
        int             nextyear;

        lastyear = nextyear = thisyear;
        if ((lastmonth = thismonth - 1) == 0) {
            lastmonth = 12;
            lastyear--;
        }
        if ((nextmonth = thismonth + 1) == 13) {
            nextmonth = 1;
```

```
            nextyear++;
        }
        printf("%9s%5d%18s%5d%18s%5d\n",
                monthname[lastmonth], lastyear,
                monthname[thismonth], thisyear,
                monthname[nextmonth], nextyear);

        printf("%s   %s   %s\n", weekday, weekday, weekday);
        calendar(lastyear, lastmonth, 0);
        calendar(thisyear, thismonth, 1);
        calendar(nextyear, nextmonth, 2);
        output(3);
#if MONTHS_PER_LINE != 3
        << error, the above won't work >>
#endif
        printf("\n\n\n");
}

output(nmonths)
int             nmonths;                /* Number of months to do    */
/*
 * Clean up and output the text.
 */
{
        register int    week;
        register int    month;
        register char   *outp;

        for (week = 0; week < WEEKS_PER_MONTH; week++) {
            outp = outline;
            for (month = 0; month < nmonths; month++) {
                /*
                 * The -1 in the following removes
                 * the unwanted leading blank from
                 * the entry for Sunday.
                 */
                sprintf(outp, "%.*s%*s",
                    DAYS_PER_WEEK * ENTRY_SIZE - 1,
                    &layout[month][week][0][1],
                    MONTH_SPACE, "");
                outp += (DAYS_PER_WEEK * ENTRY_SIZE) + MONTH_SPACE -
1;
            }
            while (outp > outline && outp[-1] == ' ')
                outp--;
            *outp = EOS;
            puts(outline);
        }
}
```

```
calendar(year, month, index)
int             year;
int             month;
int             index;          /* Which of the three months          */
/*
 * Actually build the calendar for this month.
 */

{
        register char   *tp;
        int             week;
        register int    wday;
        register int    today;

        setmonth(year, month);
        for (week = 0; week < WEEKS_PER_MONTH; week++) {
           for (wday = 0; wday < DAYS_PER_WEEK; wday++) {
                tp = &layout[index][week][wday][0];
                *tp++ = ' ';
                today = getdate(week, wday);
                if (today <= 0) {
                   *tp++ = ' ';
                   *tp++ = ' ';
                }
                else if (today < 10) {
                   *tp++ = ' ';
                   *tp   = (today + '0');
                }
                else {
                   *tp++ = (today / 10) + '0';
                   *tp   = (today % 10) + '0';
                }
           }
        }
}

usage(what, value)
char            *what;
int             value;
/*
 * Fatal parameter error
 */
{
        fprintf(stderr, "Calendar parameter error: bad %s: %d\n",
           what, value);
        fprintf(stderr, "Usage: \"calend month\" or \"calend year month\"\n");
        fprintf(stderr, "Year and month are integers.\n");
        exit(IO_ERROR);
}
```

```
/*
 * Calendar routines, intended for eventual porting to TeX
 *
 * date(year, month, week, wday)
 *      Returns the date on this week (0 is first, 5 last possible)
 *      and day of the week (Sunday == 0)
 *      Note: January is month 1.
 *

 * setmonth(year, month)
 *      Parameters are as above, sets getdate() for this month.
 *
 * int
 * getdate(week, wday)
 *      Parameters are as above, uses the data set by setmonth()
 */

/*
 * This structure is used to pass data between setmonth() and getdate().
 * It needs considerable expansion if the Julian->Gregorian change is
 * to be extended to other countries.
 */

static struct {
   int          feb;            /* Days in February for this month   */
   int          sept;           /* Days in September for this month */
   int          days_in_month;      /* Number of days in this month              */
   int          dow_first;      /* Day of week of the 1st day in month       */
} info;

static int day_month[] = {   /* 30 days hath September...          */
        0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};

int
date(year, month, week, wday)
int             year;           /* Calendar date being computed              */
int             month;          /* January == 1                              */
int             week;           /* Week in the month 0..5 inclusive  */
int             wday;           /* Weekday, Sunday == 0                      */
/*
 * Return the date of the month that fell on this week and weekday.
 * Return zero if it's out of range.
 */
{
        setmonth(year, month);
        return (getdate(week, wday));
}

setmonth(year, month)
```

```
int             year;                   /* Year to compute              */
int             month;                  /* Month, January is month 1 */
/*
 * Setup the parameters needed to compute this month
 * (stored in the info structure).
 */
{
        register int            i;


        if (month < 1 || month > 12) {          /* Verify caller's parameters  */
           info.days_in_month = 0;  /* Garbage flag                         */
           return;
        }
        info.dow_first = Jan1(year);  /* Day of January 1st for now         */
        info.feb = 29;                  /* Assume leap year             */
        info.sept = 30;                 /* Assume normal year                   */
        /*
         * Determine whether it's an ordinary year, a leap year
         * or the magical calendar switch year of 1752.
         */
        switch ((Jan1(year + 1) + 7 - info.dow_first) % 7) {
        case 1:                         /* Not a leap year              */
           info.feb = 28;
        case 2:                         /* Ordinary leap year           */
           break;

        default:                        /* The magical moment arrives       */
           info.sept = 19;              /* 19 days hath September    */
           break;
        }
        info.days_in_month =
            (month == 2) ? info.feb
           : (month == 9) ? info.sept
           : day_month[month];
        for (i = 1; i < month; i++) {
           switch (i) {             /* Special months?              */
           case 2:                      /* February                     */
                 info.dow_first += info.feb;
                 break;

           case 9:
                 info.dow_first += info.sept;
                 break;

           default:
                 info.dow_first += day_month[i];
                 break;
           }
        }
```

```
        info.dow_first %= 7;            /* Now it's Sunday to Saturday        */
}

int
getdate(week, wday)
int             week;
int             wday;
{

        register int      today;

        /*
         * Get a first guess at today's date and make sure it's in range.
         */
        today = (week * 7) + wday - info.dow_first + 1;
        if (today <= 0 || today > info.days_in_month)
           return (0);
        else if (info.sept == 19 && today >= 3)       /* The magical month?          */
           return (today + 11);         /* If so, some dates changed  */
        else                            /* Otherwise,                 */
           return (today);              /* Return the date            */
}

static int
Jan1(year)
int             year;
/*
 * Return day of the week for Jan 1 of the specified year.
 */
{
        register int      day;

        day = year + 4 + ((year + 3) / 4);      /* Julian Calendar      */
        if (year > 1800) {                      /* If it's recent, do   */
           day -= ((year - 1701) / 100);        /* Clavian correction   */
           day += ((year - 1601) / 400);        /* Gregorian correction       */
        }
        if (year > 1752)                        /* Adjust for Gregorian       */
           day += 3;                            /* calendar             */
        return (day % 7);
}
```

Apêndice D:

Descrição da funcionalidade, perfil operacional e linhas de código do Programa Cal

PROGRAMA CAL

Este programa gera o calendário de um mês ou ano especifico na tela.

Os parâmetros de entrada neste programa são mês e ano.

Argumentos:

cal – imprime mês corrente

cal x – imprime calendário do ano x

cal x y – imprime calendário do mês x, ano y

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo:

95% dos Casos de Testes Requeridoss requeridossão válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa

Dos casos válidos:

1% não possui parâmetros

49% possuem apenas um parâmetro

50% possuem dois parâmetros

5% dos casos são não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado.

25% com um único parâmetro (ano) não válido

25% com dois parâmetros, mês não válido

25% com dois parâmetros, ano não válido

25% com dois parâmetros, ambos não válidos

LINHAS DE CÓDIGO

```
/************************************************************
 *
 *           Calendar Generation Program
 *         Copyright 1986 David H. Brierley
 *
 * Permission is granted to anyone to use this software for any
 * purpose on any computer system, and to redistribute it freely,
 * subject to the following restrictions:
 * 1. The author is not responsible for the consequences of use of
 *      this software, no matter how awful, even if they arise
 *      from defects in it.
 * 2. The origin of this software must not be misrepresented, either
 *      by explicit claim or by omission.
 * 3. Altered versions must be plainly marked as such, and must not
 *      be misrepresented as being the original software.
 *
 * David H. Brierley
 * Portsmouth, RI
 * {allegra,ihnp4,linus}!rayssd!dhb
 *
 ************************************************************/

#include <stdio.h>

#define TRUE        1
#define FALSE       0

#ifndef PICDATA
#define PICDATA "/staff/dhb/data/picdata"
#endif

struct holiday_data
{
   int     h_month;
   int     h_day;
   char    h_name[14];
```

```
   struct holiday_data *h_link;
};
typedef struct holiday_data HDATA;

int     month1;
int     year1;
int     month2;
int     year2;
int     this_day;
int     this_year;
int     this_month;


int     copies;
int     points[13];
int     days[42];
int     julian_days[42];
int     first;
int     rawdata;
int     daycnt[] =
{
   31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};
int     pic_flag;
int     std_flag;
int     hday_flag;

int     easter[100] =
{
   3, 25, 4, 13, 4, 5, 4, 18, 4, 10, 4, 1, 4, 21, 4, 6, 3, 29, 4, 17,
   4, 2, 4, 22, 4, 14, 3, 29, 4, 18, 4, 10, 3, 26, 4, 14, 4, 6, 3, 29,
   4, 11, 4, 2, 4, 22, 4, 14, 3, 30, 4, 18, 4, 10, 3, 26, 4, 15, 4, 6,
   4, 19, 4, 11, 4, 3, 4, 22, 4, 7, 3, 30, 4, 19, 4, 3, 3, 26, 4, 15,
   3, 31, 4, 19, 4, 11, 4, 3, 4, 16, 4, 7, 3, 30, 4, 12, 4, 4, 4, 23
};

char    sday_file[1024];
char    hday_file[1024];
char    output_file[1024];
char    input_file[1024];

char  **line;
char   *monames[12][7];
char   *nums[10][5];
char   *holidays[12][32];
char   *special[4][12][32];

HDATA * hday_head = NULL;

FILE   *calout;
```

```
extern  FILE   *popen ();
extern  char   *index ();
extern  char   *strcpy ();
extern  char   *strncpy ();
extern  char   *gets ();
extern  char   *calloc ();
extern  char   *malloc ();
extern  char   *optarg;
extern  int    optind;

#define alloc_3d(base, x, y, z, size) {\
   register unsigned  total; register int i, j, k; register char *area;\

   total = (x * y * z) * size; area = malloc (total);\
   if (area == NULL) { perror ("calndr"); exit (1); }\
   for (i = 0; i < x; i++) for (j = 0; j < y; j++) for (k = 0; k < z; k++) { base[i][j][k] =
area; area += size; }\
}

#define alloc_2d(base, x, y, size) {\
   register unsigned  total; register int i, j; register char   *area;\
   total = (x * y) * size; area = malloc (total);\
   if (area == NULL) { perror ("calndr"); exit (1); }\
   for (i = 0; i < x; i++) for (j = 0; j < y; j++) { base[i][j] = area; area += size; }\
}

#define alloc_1d(base, x, size) {\
   register unsigned total; register int i; register char *area;\
   total = x * size; area = malloc (total);\
   if (area == NULL) { perror ("calndr"); exit (1); }\
   for (i = 0; i < x; i++) { base[i] = area; area += size; }\
}

#define zap3(base, x, y, z) {\
   register int i, j, k;\
   for (i = 0; i < x; i++) for (j = 0; j < y; j++) for (k = 0; k < z; k++) *base[i][j][k] =
'\0';\
}

#define zap2(base, x, y) {\
   register int i, j;\
   for (i = 0; i < x; i++) for (j = 0; j < y; j++) *base[i][j] = '\0';\
}

main (argc, argv)
int    argc;
char   *argv[];
{
   int    optch;
```

```
int     rc;

calout = NULL;
rawdata = 0;
(void) strcpy (input_file, PICDATA);
while ((optch = getopt (argc, argv, "r:f:o:P:")) != EOF) {
    switch (optch) {
    case 'r':
       (void) sprintf (input_file, "pic_h2m %s /tmp/picdata%d",
                        optarg, getpid ());
       fprintf (stderr, "Please wait while raw data file is transformed\n");
       rc = system (input_file);
       if (rc != 0) {
            fprintf (stderr, "Unable to transform raw data file\n");

            exit (1);
       }
       (void) sprintf (input_file, "/tmp/picdata%d", getpid ());
       rawdata = 1;
       break;
    case 'f':
       (void) strcpy (input_file, optarg);
       rawdata = 0;
       break;
    case 'o':
       calout = fopen (optarg, "w");
       if (calout == NULL) {
            fprintf (stderr, "Unable to open output file %s\n", optarg);
            exit (1);
       }
       break;
    case 'P':
       (void) sprintf (output_file, "lpr -P%s", optarg);
       calout = popen (output_file, "w");
       if (calout == NULL) {
            fprintf (stderr, "Unable to open pipe to printer %s\n", optarg);
            exit (1);
       }
       break;
    default:
       exit (1);
    }
}
if (calout == NULL) {
    fprintf (stderr, "Usage: %s [-r rawdatafile] [-f datafile]\n", argv[0]);
    fprintf (stderr, "       [-o outputfile] [-P printer]\n");
    exit (1);
}

alloc_2d (monames, 12, 7, 81);
```

```
    alloc_2d (nums, 10, 5, 6);
    alloc_2d (holidays, 12, 32, 14);
    alloc_3d (special, 4, 12, 32, 17);
    read_picdata ();

    while (cardscan () == TRUE) {
        while (copies-- > 0) {
            make_calndr ();
        }
    }

    if (rawdata == 1) {
        (void) unlink (input_file);
    }


}

make_calndr ()
{
    int     i;
    int     j;

    this_year = year1;
    this_month = month1 - 1;
    daycnt[1] = 28 + leap ();
    gen_hdays ();
    this_day = 0;
    for (i = 0; i < this_month; i++) {
        this_day += daycnt[i];
    }

    while (this_year < year2 || (this_year == year2 && this_month < month2)) {
        this_month++;
        if (this_month < 1) {
            this_month = 1;
        }
        if (this_month > 12) {
            this_month = 1;
            this_year++;
            this_day = 0;
            daycnt[1] = 28 + leap ();
            gen_hdays ();
        }
        if (pic_flag) {
            print_picture (this_month);
        }
        caltop ();
        first = zeller (this_month, this_year);
        for (i = 0; i < 42; i++) {
```

```
            days[i] = 0;
            julian_days[i] = 0;
        }
        for (i = 0; i < daycnt[this_month - 1]; i++) {
            days[i + first] = i + 1;
            julian_days[i + first] = (i + 1) + this_day;
        }
        this_day += daycnt[this_month - 1];
        dash_line (1);

        for (i = 0; i < 6; i++) {
            vbar_line ();
            date_line (i);
            for (j = 0; j < 4; j++) {
                special_line (j, this_month, i);
            }

            jdate_line (this_month, i);
            dash_line (0);
        }
    }
}

leap ()
{
    int     yr;

    yr = this_year;

    if ((yr / 4) * 4 != yr) {
        return 0;
    }
    if ((yr / 100) * 100 != yr) {
        return 1;
    }
    if ((yr / 400) * 400 != yr) {
        return 0;
    }
    return 1;

}

zeller (month, year)
int     month;
int     year;
{
    int     result;
    int     m,
            c,
            y,
```

```
        f;

    y = year;
    m = month - 2;
    if (m <= 0) {
        m += 12;
        y--;
    }

    c = y / 100;
    y -= (c * 100);

    f = ((26 * m - 2) / 10) + 1 + ((5 * y) / 4) + (c / 4) - (2 * c);

    result = f % 7;
    return result;


}

cardscan ()
{
    char    input_line[80];
    char   *ptr;
    int     plus_minus;
    int     n;

    if (isatty (0)) {
        printf ("Enter command line, enter '?' for help, 'q' to quit\n");
        printf (">> ");
        (void) fflush (stdout);
    }

    if (getline (0, input_line) == -1) {
        if (isatty (0)) {
            printf ("\n");
        }
        return FALSE;
    }

    if (input_line[0] == 'q') {
        return FALSE;
    }

    if (input_line[0] == '?') {
        help ();
        return cardscan ();
    }

    n = sscanf (input_line, "%d/%d,%d/%d", &month1, &year1, &month2, &year2);
```

```
if (n != 4) {
    printf ("Error: badly formed input line\n");
    return cardscan ();
}
ptr = index (input_line, ' ');
ptr++;
zap3 (special, 4, 12, 32);
plus_minus = TRUE;
copies = 1;
pic_flag = TRUE;
std_flag = TRUE;
hday_flag = TRUE;
(void) strcpy (sday_file, "");

while (*ptr) {
    switch (*ptr) {
    case '+':
       plus_minus = TRUE;

       break;
    case '-':
       plus_minus = FALSE;
       break;
    case 'p':
       pic_flag = plus_minus;
       break;
    case 'h':
       hday_flag = plus_minus;
       if (hday_flag) {
            read_holidays (ptr + 1);
       }
       if (hday_file[0] == '\0') {
            hday_flag = FALSE;
       }
       ptr = index (ptr, ' ');
       break;
    case 's':
       std_flag = plus_minus;
       break;
    case 'x':
       if (plus_minus) {
            read_sdays (ptr + 1);
       }
       ptr = index (ptr, ' ');
       break;
    case 'c':
       copies = atoi (ptr + 1);
       if (copies < 1) {
            copies = 1;
       }
```

```
            ptr = index (ptr, ' ');
            break;
        }
        ptr++;
    }

    printf ("options: begin = %02d/%d, end = %02d/%d, copies = %d\n",
            month1, year1, month2, year2, copies);

    if (pic_flag) {
        printf ("\t\tpictures");
    }
    else {
        printf ("\t\tnopictures");
    }

    if (std_flag) {
        printf (", standard");
    }

    else {
        printf (", nostandard");
    }

    if (hday_flag) {
        printf (", holidays(%s)", hday_file);
    }
    else {
        printf (", noholidays");
    }

    if (sday_file[0] != '\0') {
        printf (", specialdays(%s)", sday_file);
    }
    else {
        printf (", nospecialdays");
    }

    printf ("\n");

    return TRUE;
}

read_sdays (ptr)
char   *ptr;
{
    char    sd_line[256];
    char   *line_ptr;
    char   *msg;
    int     fd_sday;
```

```
int     sp_indx;
int     month;
int     day;
int     i,
        j,
        k,
        n;

(void) strcpy (sd_line, ptr);
msg = index (sd_line, ' ');
if (msg != NULL) {
    *msg = '\0';
}
fd_sday = open (sd_line, 0);
if (fd_sday == -1) {
    return;
}
(void) strcpy (sday_file, sd_line);

while (getline (fd_sday, sd_line) != -1) {

    line_ptr = sd_line;
    while (*line_ptr && (*line_ptr == ' ')) {
        line_ptr++;
    }
    if (!*line_ptr) {
        continue;
    }
    n = sscanf (line_ptr, "%d%*c%d%*c%d", &i, &j, &k);
    switch (n) {
    case 3:
        if ((i < 1) || (i > 4)) {
                printf ("ignoring special day line '%s'\n", sd_line);
                continue;
        }
        if (i == 4) {
                printf ("Warning: special day '%s' ", sd_line);
                printf ("might be overwritten by two line holiday\n");
        }
        sp_indx = i - 1;
        month = j - 1;
        day = k;
        break;
    case 2:
        sp_indx = 1;
        month = i - 1;
        day = j;
        break;
    default:
        printf ("ignoring special day line '%s'\n", sd_line);
```

```
            continue;
        }
        if (*special[sp_indx][month][day] != '\0') {
            printf ("Warning: special day conflicts with previous entry\n");
            printf ("currently processing file %s\n", sday_file);
            printf ("input line = '%s'\n", sd_line);
            printf ("previous entry = '%s'\n", special[sp_indx][month][day]);
        }
        msg = index (line_ptr, ' ');
        while (*msg == ' ') {
            msg++;
        }
        (void) strcpy (special[sp_indx][month][day], msg);
    }
    (void) close (fd_sday);
}

special_line (indx, month, week)
int     indx;
int     month;
int     week;

{
    int     day;

    fprintf (calout, "  ");
    month--;
    for (day = week * 7; day < (week + 1) * 7; day++) {
        fprintf (calout, "| %-16s", special[indx][month][days[day]]);
    }
    fprintf (calout, "|\n");
}

date_line (week)
int     week;
{
    int     day;

    fprintf (calout, "  |");
    for (day = week * 7; day < (week + 1) * 7; day++) {
        if (days[day] != 0) {
            fprintf (calout, "%16d |", days[day]);
        }
        else {
            fprintf (calout, "%16s |", " ");
        }
    }
    fprintf (calout, "\n");
}
```

```
jdate_line (month, week)
int     month;
int     week;
{
    int     day;

    fprintf (calout, "  ");
    month--;
    for (day = week * 7; day < (week + 1) * 7; day++) {
        if (days[day] != 0) {
            fprintf (calout, "|%03d %-13s", julian_days[day],
                    holidays[month][days[day]]);
        }
        else {
            fprintf (calout, "|%17s", " ");
        }
    }
    fprintf (calout, "|\n");
}

dash_line (n)
int     n;

{
    while (n-- > 0) {
        fprintf (calout, "\n");
    }

    fprintf (calout, "  ");
    for (n = 0; n < 7; n++) {
        fprintf (calout, "+----------------");
    }
    fprintf (calout, "+\n");
}

vbar_line ()
{
    int     n;

    fprintf (calout, "  ");
    for (n = 0; n < 7; n++) {
        fprintf (calout, "%-18s", "|");
    }
    fprintf (calout, "|\n");
}

caltop ()
{
    int     i;
    int     y_thou;
```

```
    int     y_hund;
    int     y_ten;
    int     y_one;

    y_thou = this_year / 1000;
    y_hund = (this_year % 1000) / 100;
    y_ten = (this_year % 100) / 10;
    y_one = (this_year % 10);

    fprintf (calout, "\f");

    for (i = 0; i < 8; i++) {
        switch (i) {
        case 0:
        case 1:
            fprintf (calout, "%26s%s\n", " ", monames[this_month - 1][i]);
            break;
        case 2:
            fprintf (calout, "%6s%-20s%-94s%s\n", " ",
                    nums[y_hund][i - 2], monames[this_month - 1][i],
                    nums[y_ten][i - 2]);
            break;
        default:

          fprintf (calout, "%-6s%-20s%-94s%-6s%s\n",
                    nums[y_thou][i - 3], nums[y_hund][i - 2],
                    monames[this_month - 1][i],
                    nums[y_ten][i - 2], nums[y_one][i - 3]);
            break;
        case 7:
            fprintf (calout, "%-126s%s\n",
                    nums[y_thou][i - 3], nums[y_one][i - 3]);
            break;
        }
    }
    fprintf (calout, "\n%-9s%-18s%-18s%-18s%-18s%-18s%-18s%s\n",
            " ", "Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday");

}



print_picture (month)
int     month;
{
    int     i;
    char   *outline;

    for (i = points[month - 1]; i < points[month]; i++) {
```

```
        outline = line[i];
        switch (*outline) {
        case '1':
            fprintf (calout, "\f");
            break;
        case '-':
            fprintf (calout, "\n\n\n");
            break;
        case '0':
            fprintf (calout, "\n\n");
            break;
        case ' ':
            fprintf (calout, "\n");
            break;
        case '+':
            fprintf (calout, "\r");
            break;
        default:
            fprintf (calout, "\n");
            break;
        }
        fprintf (calout, "%s", ++outline);
    }
    fprintf (calout, "\r");

}

read_picdata ()
{
    int     picdata;
    unsigned    numlines;

    picdata = open (input_file, 0);
    if (picdata == -1) {
        perror ("calndr");
        fprintf (stderr, "Unable to open data file\n");
        exit (1);
    }

    if (read (picdata, monames[0][0], 12 * 7 * 81) != (12 * 7 * 81)) {
        fprintf (stderr, "Incomplete read on data file while reading sss\n");
        exit (1);
    }
    if (read (picdata, nums[0][0], 10 * 5 * 6) != (10 * 5 * 6)) {
        fprintf (stderr, "Incomplete read on data file while reading sss\n");
        exit (1);
    }
    if (read (picdata, (char *) points, sizeof points) != sizeof points) {
        fprintf (stderr, "Incomplete read on data file while reading sss\n");
        exit (1);
```

```
    }
    numlines = (unsigned) points[12];
    line = (char **) calloc (numlines, sizeof (char *));
    alloc_1d (line, numlines, 134);
    if (read (picdata, line[0], (int) (numlines * 134)) != (numlines * 134)) {
        fprintf (stderr, "Incomplete read on data file while reading sss\n");
        exit (1);
    }
    (void) close (picdata);
    return;

}

/*----- subroutine get_line                                    -----*/

#define BUFFSIZE 4096
#define BUFCOUNT 6

getline (fd, buf)
int     fd;
char   *buf;
{

    static char buffer[BUFCOUNT][BUFFSIZE];
    static int  buf_pos[BUFCOUNT];

    static int  buf_size[BUFCOUNT];
    static int  init_flag = 1;
    int     i;


    if (init_flag) {
        for (i = 0; i < BUFCOUNT; i++) {
            buf_pos[i] = BUFFSIZE;
            buf_size[i] = BUFFSIZE;
        }
        init_flag = 0;
    }

    if (fd >= BUFCOUNT) {
        printf ("Error: get_line: fd = %d\n", fd);
        exit (1);
    }


    i = 0;

    while (1) {
        if (buf_pos[fd] >= buf_size[fd]) {
            buf_size[fd] = read (fd, buffer[fd], BUFFSIZE);
```

```
            buf_pos[fd] = 0;
            if (buf_size[fd] == 0) {
                break;
            }
        }

        if (buffer[fd][buf_pos[fd]] == '\n')
            break;
        if (buffer[fd][buf_pos[fd]] == 12) {
            buf_pos[fd]++;
            continue;
        }
        *buf = buffer[fd][buf_pos[fd]++];
        if (*buf == '\t') {
            *buf = ' ';
            while (i % 8 != 7) {
                *++buf = ' ';
                i++;
            }
        }
        buf++;
        i++;

    }

    *buf = '\0';

    if (i == 0 && buf_size[fd] == 0 && buf_pos[fd] == 0) {
        i = -1;
    }
    buf_pos[fd]++;

    return (i);

}

#define         VU(s)  fprintf(vu, s)

help ()
{
    FILE   *vu;
    FILE   *popen ();
    char    junk[256];

#ifdef PAGER
    if ((vu = popen (PAGER, "w")) == NULL)
#endif
        vu = stdout;

    VU ("The format of a command input line is as follows:\n");
```

```
    VU (" mm/yyyy,mm/yyyy [options]\n\n");
    VU ("The first mm/yyyy specifies the month and year the\n");
    VU ("calendar is to begin with and the second mm/yyyy specifies\n");
    VU ("the ending month and year.  The options that may be given\n");
    VU ("are as follows:\n");
    VU ("\n");
    VU ("   +p       = print pictures\n");
    VU ("   -p       = dont print pictures\n");
    VU ("   +hFILE   = read list of special holidays from FILE\n");
    VU ("   -h       = no special holidays\n");
    VU ("   +s       = include standard holidays on calendar\n");
    VU ("   -s       = dont include standard holidays\n");
    VU ("   +xFILE   = read list of special days from FILE\n");
    VU ("   +cNN     = print NN copies of calendar\n");
    VU ("\n");
#ifndef PAGER
    VU ("Type RETURN for more help\n");
    (void) getline (0, junk);
#endif
    VU ("The special day notations are written on the four lines\n");
    VU ("in the center of the block for the specified day.\n");
    VU ("The format of the special day entries is as follows:\n");
    VU ("\n");
    VU ("   n:mm/dd descriptive text\n");
    VU ("\n");
    VU ("The 'n:' at the beginning of the line specifies which\n");
     VU ("of the four lines in the block to print the descriptive\n");

    VU ("text on.  If the 'n:' is omitted, the default is to print\n");
    VU ("the text on the third line.  The format of the holiday\n");
    VU ("entries is the same except that the 'n:' is always omitted.\n");
    VU ("The text of a holiday entry is printed on the bottom line\n");
    VU ("of the block, next to the julian date.\n\n");
    VU ("The descriptive text can be up to 16 characters for a\n");
    VU ("special day entry and up to 13 characters for a holiday.\n");

    if (vu != stdout) {
        (void) pclose (vu);
    }

}

gen_hdays ()
{
    HDATA * ptr;

    if ((this_year == year2) && (month2 == 1)) {
        return;
    }
```

```
    zap2 (holidays, 12, 32);

    if (std_flag) {
        std_hdays ();
    }

    if (!hday_flag) {
        return;
    }
    if (hday_head == NULL) {
        return;
    }

    ptr = hday_head;
    while (ptr) {
        if (*holidays[ptr -> h_month][ptr -> h_day] != '\0') {
            printf ("User specified holiday conflicts with standard holiday\n");
            printf ("\tstandard holiday = '%s', user holiday = '%s'\n",
                    holidays[ptr -> h_month][ptr -> h_day], ptr -> h_name);
            (void) fflush (stdout);
        }
        (void) strcpy (holidays[ptr -> h_month][ptr -> h_day], ptr -> h_name);
        ptr = ptr -> h_link;
    }

}

std_hdays ()

{
    int     temp;
    int     month;

    /* fixed holidays */
    (void) strcpy (holidays[0][1], "New Years Day");
    (void) strcpy (special[3][0][15], "   Martin Luther");
    (void) strcpy (holidays[0][15], " King Jr. Day");
    (void) strcpy (holidays[1][2], "Groundhog Day");
    (void) strcpy (holidays[1][14], "Valentine Day");
    (void) strcpy (special[3][1][12], "       Lincoln's");
    (void) strcpy (holidays[1][12], "    Birthday ");
    (void) strcpy (special[3][1][22], "    Washington's");
    (void) strcpy (holidays[1][22], "   Birthday  ");
    (void) strcpy (special[3][2][17], " Saint Patrick's");
    (void) strcpy (holidays[2][17], "    Day      ");
    (void) strcpy (holidays[5][14], "Flag Day     ");
    (void) strcpy (special[3][6][4], "     Independence");
    (void) strcpy (holidays[6][4], "      Day    ");
    (void) strcpy (special[3][9][24], "  United Nations");
    (void) strcpy (holidays[9][24], "    Day      ");
```

```
(void) strcpy (holidays[9][31], "Halloween    ");
(void) strcpy (holidays[10][11], "Veterans Day ");
(void) strcpy (holidays[11][25], "Christmas    ");

/* Armed Forces Day */
temp = 21 - zeller (5, this_year);
(void) strcpy (special[3][4][temp], "    Armed Forces");
(void) strcpy (holidays[4][temp], "     Day     ");

/* Labor Day */
temp = 9 - zeller (9, this_year);
if (temp > 7) {
    temp -= 7;
}
(void) strcpy (holidays[8][temp], "Labor Day");

/* Thanksgiving */
temp = 26 - zeller (11, this_year);
if (temp < 22) {
    temp += 7;
}
(void) strcpy (holidays[10][temp], "Thanksgiving");

/* Memorial Day */
temp = 30 - zeller (5, this_year);
if (temp < 25) {
    temp += 7;
}
(void) strcpy (holidays[4][temp], "Memorial Day");

/* Mothers Day */
temp = 15 - zeller (5, this_year);
if (temp > 14) {
    temp -= 7;
}
(void) strcpy (holidays[4][temp], "Mother's Day");

/* Fathers Day */
temp = 22 - zeller (6, this_year);
if (temp > 21) {
    temp -= 7;
}
(void) strcpy (holidays[5][temp], "Father's Day");

/* Columbus Day */
temp = 16 - zeller (10, this_year);
if (temp > 12) {
    temp -= 7;
}
(void) strcpy (holidays[9][temp], "Columbus Day");
```

```
/* generate a couple of Canadian holidays */
temp = 23 - zeller (5, this_year);
(void) strcpy (special[3][4][temp], "     Canadian   ");
(void) strcpy (holidays[4][temp], "Victoria Day ");

temp = zeller (7, this_year);
switch (temp) {
case 0:
    temp = 2;
    break;
case 6:
    temp = 3;
    break;
default:
    temp = 1;
    break;
}
(void) strcpy (special[3][6][temp], "     Canadian   ");
(void) strcpy (holidays[6][temp], "Dominion Day ");

/* if year is withing range, generate easter etc. */
temp = this_year - 1950;
if ((temp < 1) || (temp > 50)) {
    printf ("Warning: requested year is not within the bounds for\n");
    printf ("generating Easter and related holidays.\n");
    return;
}



temp--;
temp *= 2;
month = easter[temp];
temp = easter[++temp];
(void) strcpy (holidays[--month][temp], "Easter");

temp -= 2;
if (temp < 1) {
    month--;
    temp += daycnt[month];
}
(void) strcpy (holidays[month][temp], "Good Friday");

temp -= 5;
if (temp < 1) {
    month--;
    temp += daycnt[month];
}
(void) strcpy (holidays[month][temp], "Palm Sunday");
```

```
    temp -= 39;
    while (temp < 1) {
        month--;
        temp += daycnt[month];
    }
    (void) strcpy (holidays[month][temp], "Ash Wednesday");

}

read_holidays (sptr)
char   *sptr;
{
    char    file[1024];
    char    input_line[80];
    char   *temp;
    int     fd;
    int     i,
            j,
            n;
    HDATA  *ptr;
    HDATA  *next;

    if (*sptr == ' ') {
        return;
    }

    if (hday_head != NULL) {
        ptr = hday_head;
        while (ptr) {
            next = ptr -> h_link;

             free ((char *) ptr);
            ptr = next;
        }
    }

    hday_head = NULL;
    (void) strcpy (hday_file, "");
    (void) strcpy (file, sptr);
    if ((temp = index (file, ' ')) != NULL) {
        *temp = '\0';
    }

    if ((fd = open (file, 0)) == -1) {
        printf ("** warning ** - unable to open file %s\n", file);
        printf ("               +h option ignored\n");
        hday_flag = FALSE;
        return;
    }
```

```
    (void) strcpy (hday_file, file);

    while (getline (fd, input_line) != -1) {
        temp = input_line;
        while (*temp && (*temp == ' ')) {
            temp++;
        }
        if (!*temp) {
            continue;
        }
        n = sscanf (temp, "%d%*c%d", &i, &j);
        if (n != 2) {
            continue;
        }
        next = (HDATA *) malloc (sizeof (HDATA));
        if (next == NULL) {
            perror ("calndr");
            printf ("Unable to alocate space for holiday_data\n");
            return;
        }
        if (hday_head == NULL) {
            hday_head = next;
        }
        else {
            ptr -> h_link = next;
        }
        ptr = next;
        ptr -> h_link = NULL;
        i--;
        temp = index (temp, ' ');
        while (*temp && (*temp == ' ')) {
            temp++;

}
        ptr -> h_month = i;
        ptr -> h_day = j;
        (void) strncpy (ptr -> h_name, temp, 13);
    }

    (void) close (fd);

}
```

Apêndice E:

Descrição da funcionalidade, perfil operacional e linhas de código do programa Jday-Jdate

PROGRAMA JDAY-JDATE

O programa aceita uma data como parâmetro e retorna o dia juliano, o dia juliano é o número de dias da data fornecida como parâmetro até uma data especifica de um passado distante.

Argumentos:

jday-jdate x – calcula a data Gregoriana correspondente ao número Juliano x

jday-jdate d m y – calcula o número correspondente à data

Entradas válidas:

1.º parâmetro: de 1 até 12

2.º parâmetro: de 1 até 28 (fevereiro) ou 1 até 29 (fevereiro em ano bissexto) ou 1 a 30 para os meses (abril, junho, setembro, novembro) ou de 1 a 31 (janeiro, março, maio, julho, agosto, outubro, dezembro)

3.º parâmetro: de 0 até 522485619.

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo:

95% dos Casos de Testes Requeridoss requeridossão válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa

Dos casos válidos:

50% terão o formato mês dia e ano mm dd aa

50% terão um número do calendário Juliano

5% dos casos são não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado

Dos casos não válidos:

1% sem parâmetro

19% com um parâmetro

19% com dois parâmetros

19% com três parâmetros, dois válidos e um inválido

19% com três parâmetros, um válido e dois inválidos

19% com os três parâmetros inválidos

4% com o número juliano inválido

LINHAS DE CÓDIGO

```
5-Jul-85 08:21:39-MDT,2308;000000000001
Return-Path: <unix-sources-request@BRL.ARPA>
Received: from BRL-TGR.ARPA by SIMTEL20.ARPA with TCP; Fri 5 Jul 85
08:18:08-MDT
Received: from usenet by TGR.BRL.ARPA id a027771; 5 Jul 85 9:46 EDT
From: nrh@inmet.uucp
Newsgroups: net.sources
Subject: Re: date <-> day-number software sought
Message-ID: <12900006@inmet.UUCP>
Date: 3 Jul 85 14:06:00 GMT
Nf-ID: #R:bocklin:-23200:inmet:12900006:000:1746
Nf-From: inmet!nrh  Jul  3 10:06:00 1985
To:     unix-sources@BRL-TGR.ARPA


Here are two routines, jday.c and jdate.c that have served
me pretty well.  They are translations from ALGOL in Collected
Algorithms of CACM.
/*
** Takes a date, and returns a Julian day. A Julian day is the number of
** days since some base date  (in the very distant past).
** Handy for getting date of x number of days after a given Julian date
** (use jdate to get that from the Gregorian date).
** Author: Robert G. Tantzen, translator: Nat Howard
** Translated from the algol original in Collected Algorithms of CACM
** (This and jdate are algorithm 199).
*/
long
jday(mon, day, year)
int mon, day, year;
{
 long m = mon, d = day, y = year;
 long c, ya, j;
```

```
if(m > 2) m -= 3;
else {
        m += 9;
        --y;
}
c = y/100L;
ya = y - (100L * c);
j = (146097L * c) /4L + (1461L * ya) / 4L + (153L * m + 2L)/5L + d + 1721119L;
return(j);
}
/* Julian date converter. Takes a julian date (the number of days since
** some distant epoch or other), and returns an int pointer to static space.
** ip[0] = month;
** ip[1] = day of month;
** ip[2] = year (actual year, like 1977, not 77 unless it was  77 a.d.);
** ip[3] = day of week (0->Sunday to 6->Saturday)
** These are Gregorian.
** Copied from Algorithm 199 in Collected algorithms of the CACM
** Author: Robert G. Tantzen, Translator: Nat Howard
*/
int *
jdate(j)
long j;
{
 static int ret[4];

 long d, m, y;

 ret[3] = (j + 1L)%7L;
 j -= 1721119L;
 y = (4L * j - 1L)/146097L;
 j = 4L * j - 1L - 146097L * y;
 d = j/4L;
 j = (4L * d + 3L)/1461L;
 d = 4L * d + 3L - 1461L * j;
 d = (d + 4L)/4L;
 m = (5L * d - 3L)/153L;
 d = 5L * d - 3 - 153L * m;
 d = (d + 5L) / 5L;
 y = 100L * y + j;
 if(m < 10)
        m += 3;
 else {
        m -= 9;
        ++y;
 }
 ret[0] =  m;
 ret[1] = d;
```

```
 ret[2] = y;


 return(ret);
}
```

Apêndice F:

Descrição da funcionalidade, perfil operacional e linhas de código do Programa *Dateplus*

PROGRAMA *DATEPLUS*

O programa realiza operações de soma e subtração entre uma data fornecida como parâmetro e outros parâmetros que foram introduzidos no momento da execução.

Argumentos:

- *Dateplus* x y unidade – calcula a data representada pelo número x, incrementada de y unidades
- as unidades podem ser: sec, min hour, day, week, mon, year

PERFIL OPERACIONAL

O perfil operacional do script utilizado para gerar os Casos de Testes Requeridoss requeridospara esse programa foi definido como sendo:

95% dos Casos de Testes Requeridoss requeridossão válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosestão dentro dos limites de cada parâmetro do programa

Dos casos válidos:

15% são no formato mmddaa operação + valor unidade
15% são no formato mmddaa mais 2 vezes (operação + valor unidade)
15% são no formato mmddaa mais 3 vezes (operação + valor unidade)
15% são no formato mmddaa mais 4 vezes (operação + valor unidade)
15% são no formato mmddaa mais 5 vezes (operação + valor unidade)
15% são no formato mmddaa mais 6 vezes (operação + valor unidade)
10% são no formato mmddaa mais 7 vezes (operação + valor unidade)

Obs:
operação - pode ser a subtração (-) ou a soma (+)
valor - é a quantidade que será incrementada ou decrementada da data
unidade - é o tipo de unidade utilizada para especificar o valor (sec, min hour, day, week, mon, year)

5% dos casos são não válidos, representa que os valores de entrada utilizados como Casos de Testes Requeridoss requeridosnão estão dentro dos limites de cada parâmetro do programa ou não atendem algum requisito especificado

Dos casos não válidos:

- 50% com a data válida, mas com pelo menos uma opção inválida (operação + valor unidade), podendo ser da seguinte maneira:

  5% sem a data

  15% somente com a data

  20% com a data, operação, valor e a unidade mas com essa opção inválida

  10% com a data mais 2 vezes (operação + valor unidade) mas com uma desses opções inválidas

  10% com a data mais 3 vezes (operação + valor unidade) mas com uma desses opções inválidas

  10% com a data mais 4 vezes (operação + valor unidade) mas com uma desses opções inválidas

  10% com a data mais 5 vezes (operação + valor unidade) mas com uma desses opções inválidas

  10% com a data mais 6 vezes (operação + valor unidade) mas com uma desses opções inválidas

  10% com a data mais 7 vezes (operação + valor unidade) mas com uma desses opções inválidas

- 50% com a data inválida, podendo ter até todas as opções válidas, sendo da seguinte maneira:

  5% sem a data

  15% somente com a data

  20% com a data, operação, valor e unidade

  10% com a data mais 2 vezes (operação + valor unidade)

  10% com a data mais 3 vezes (operação + valorunidade)

  10% com a data mais 4 vezes (operação + valor unidade)

  10% com a data mais 5 vezes (operação + valor unidade)

10% com a data mais 6 vezes (operação + valor unidade)

10% com a data mais 7 vezes (operação + valor unidade)

LINHAS DE CÓDIGO

```
/* date+ - add specified time to current date */

/* Please send additions, bug fixes, portifications, etc. to:
 *
 *      Daniel LaLiberte
 *      ihnp4!uiucdcs!liberte
 *      University of Illinois, Urbana-Champaign
 *      Department of Computer Science
 */

/* This is written for BSD42 */

/* This was revised for SYSTEM V by:
 *
 *      Robert O. Domitz
 *      ...!vax135!petsd!pecnos!rod
 *      Concurrent Computer Corporation
 *      106 Apple Street
 *      Tinton Falls, NJ  07724
 */

#include <stdio.h>
#include <time.h>

long tloc;
long time();

char *ctime();
struct tm *localtime();

struct tm *ts;

double atof();


int argc;       /* global argument passing */
char **argv;



main (Argc, Argv)
```

```
/* return time + 1st arg hours */

int Argc;
char *Argv[];

{
        argc = Argc;
        argv = Argv;
        incrdate();
        printdate();

} /* main */



incrdate()      /* increment date from arguments */
{
        static char *unit[] =
        {
                "sec", "min", "hour", "day", "week", "mon", "year", ""        };

        static int conv[] =       /* conversion factor */
        {
                1, 60, 3600, 86400, 604800, 0, 0       };

        int i;
        double value;
        long total;       /* cummulative total increment in whole seconds */
        double monthincr = 0.0,         /* store increment of month and year */
        yearincr = 0.0;           /* since months and years are not uniform */

        time(&tloc);    /* current time */
        argc--;
        argv++;

        while (argc &&
           ((**argv == '.') ||
           (**argv == '-') ||
           (**argv == '+') ||
           (**argv >= '0' && **argv <= '9'))) {
                value = atof(argv[0]);
/*              printf("%s = %f", argv[0], value); */

                argv++;
                argc--;
                if (argc == 0) missing();
                else {  /* search for unit */
                        for (i = 0;       (i < 7) &&
                                (0 != strncmp(argv[0], unit[i],
                                        strlen(unit[i])));)
```

```
                        i++;
                    if (i == 7) missing();
                    else {

                        argv++;
                        argc--;
                        if (i < 5) value *= conv[i];
                        if (i == 5) monthincr += value;
                        if (i == 6) yearincr += value;
                    }
/*                  printf(" %s (%f seconds)\n", unit[i], value); */
                }

                total += value;
        }


        tloc += total;
        ts = localtime(&tloc);
        ts->tm_mon += monthincr;
        ts->tm_year += yearincr;

} /* getincr */


missing()
{
        fprintf(stderr, "date+: missing unit\n");
        exit (1);
}


printdate()
{
        char *format;

        static char *month[] =
           {
                "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"    };

        static char *day[] =
           {
                "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"    };


        if (argc == 0) {        /* put default format on argv */
                argv[0] = "%H:%M %h %d";
                argc++;
        }
```

```
        while (argc > 0) {
                format = argv[0];


                while (*format) {
                        if (*format != '%')
                                putchar(*format);
                        else if (format[1])

                                switch(*++format) {
                                case 'n':
                                        putchar ('\n');
                                        break;
                                case 't':
                                        putchar ('\t');
                                        break;
                                case 'S':
                                        printf("%02d", ts->tm_sec);
                                        break;
                                case 'Z':
                                        printf("%d", tloc);
                                        break;
                                case 'M':
                                        printf("%02d", ts->tm_min);
                                        break;
                                case 'H':
                                        printf("%02d", ts->tm_hour);
                                        break;
                                case 'T':
                                        printf("%02d:%02d:%02d",
                                                ts->tm_hour, ts->tm_min,
                                                ts->tm_sec);
                                        break;
                                case 'd':
                                        printf("%02d", ts->tm_mday);
                                        break;
/*                              case 'D':
                                        printf("%d", ts->tm_mday);
                                        break;                      */
                                case 'm':
                                        printf("%02d", ts->tm_mon + 1);
                                        break;
                                case 'h':
                                        printf("%s", month[ts->tm_mon]);
                                        break;
                                case 'y':
                                        printf("%02d", ts->tm_year);
                                        break;
                                case 'w':
```

```
                                printf("%1d", ts->tm_wday);
                                break;
                        case 'a':
                                printf("%s", day[ts->tm_wday]);
                                break;

                        case 'D':
                                printf("%02d/%02d/%02d",
                                        ts->tm_mon + 1, ts->tm_mday,
                                        ts->tm_year);
                                break;
                        case 'j':
                                printf("%d", ts->tm_yday);
                                break;

                        default:
        fprintf(stderr, "date+: Bad format character: '%c'\n", *format);
                                exit(1);
                        }

                        format++;
                } /* while (*format) */

                argc--;
                argv++;
                if (argc > 0)
                        putchar(' ');
        } /* while (argc > 0) */

        putchar('\n');
} /* printdate */
```

Apêndice G:

Conforme JOORGE *et al* (2001) a ferramenta PROTEUM/IM 2.0 conta com 75 operadores para o teste de unidade e 33 para o teste de integração.

O conjunto de operadores de mutação de unidade é classificado em quatro classes : Mutação de Constantes, Mutação de Comandos, Mutação de Operadores e Mutação de Variáveis.

Para a realizaçao dos experimentos desse trabalho, foram selecionados 65 operadores de unidade. Cada conjunto de operadores de mutação revela diferentes tipos de erros em um mesmo tipo de estrutura.

Descrição dos Operadores de Mutação (Fonte: Jorge *et al*., 2001)

Operador	Descrição
Cccr	Troca constantes por todas as constantes do programa
Ccsr	Troca referências escalares por constantes
CRCR	Troca cada referência escalar por constantes requeridas dependendo do tipo da referência
OAAA	Troca um operador aritmético com atribuições por outro operador aritmético com atribuições
OAAN	Troca um operador aritmético sem atribuições por outro operador aritmético sem atribuições
OABA	Troca um operador aritmético com atribuições por operador bitwise com atribuições
OABN	Troca um operador aritmético sem atribuições por um operador bitwise sem atribuições
OAEA	Troca um operador aritmético com atribuições por um operador plano
OALN	Troca um operador aritmético sem atribuições por um operador lógico
OARN	Troca um operador aritmético sem atribuições por um operador relacional
OASA	Troca um operador aritmético com atribuições por um operador de deslocamento com atribuições
OASN	Troca um operador aritmético sem atribuições por um operador de deslocamento sem atribuições
OBAA	Troca um operador bitwise com atribuições por um operador aritmético com atribuições
OBAN	Troca um operador bitwise sem atribuições por um operador aritmético sem atribuições
OBBA	Troca um operador bitwise com atribuições por outro

operador bitwise com atribuições

OBBN Troca um operador bitwise sem atribuições por um outro operador bitwise sem atribuições

OBEA Troca um operador bitwise com atribuições por um operador plano

OBLN Troca um operador bitwise sem atribuições por um operador lógico

OBNG Reverte o contexto das expressões comparando bitwise

OBRN Troca um operador bitwise sem atribuições por um operador relacional

OBSA Troca um operador bitwise com atribuições por um operador swift com atribuições

OBSN Troca um operador bitwise sem atribuições por um operador de deslocamento sem atribuições

OCNG Modela erros em comando de seleção e repetição, revertendo essas condições. Utilizado quando não envolve nenhum operador lógico

OCOR Troca por um operador de cast por todos os outros tipos primitivos da linguagem

OEAA Troca um operador plano por um operador aritmético com atribuições

OEBA Troca um operador plano por um operador bitwise com atribuições

OESA Troca um operador plano por um operador de deslocamento com atribuições

Oido Modela erros que surgem com o uso incorreto dos operadores ++ e - -

OIPM Revela erros no uso incorreto de expressões que envolvam ++, -- e operadores de indireção *.

OLAN Troca um operador lógico por um operador aritmético sem atribuições

OLBN Troca um operador lógico por um operador bitwise sem atribuições

OLLN Troca um operador lógico por outro operador lógico

OLNG Modela erros em comandos de seleção e repetição, revertendo essas condições

OLRN Troca um operador lógico por um operador relacional

OLSN Troca um operador lógico por um operador de deslocamento sem atribuições

ORAN Troca um operador relacional por um operador aritmético sem atribuições

ORBN Troca um operador relacional por um operador bitwise sem atribuições

ORLN Troca um operador relacional por um operador lógico

ORRN Troca um operador relacional por outro operador relacional

ORSN Troca um operador relacional por um operador de deslocamento sem atribuições

OSAA	Troca um operador de deslocamento com atribuições por um operador aritmético com atribuições
OSAN	Troca um operador de deslocamento sem atribuições por um operador aritmético sem atribuições
OSBA	Troca um operador de deslocamento com atribuições por um operador bitwise com atribuições
OSBN	Troca um operador de deslocamento sem atribuições por um operador bitwise sem atribuições
OSEA	Troca um operador de deslocamento com atribuições por um operador plano
OSLN	Troca um operador de deslocamento sem atribuições por um operador lógico
OSRN	Troca um operador de deslocamento sem atribuições por um operador relacional
OSSA	Troca um operador de deslocamento com atribuições por outro operador de deslocamento com atribuições
OSSN	Troca um operador de deslocamento sem atribuições por outro operador de deslocamento sem atribuições
SBRC	Troca os comandos *break* por comandos *continue*
SCRB	Troca os comandos *continue* por comandos *break*
SGLR	Troca os rótulos dos comandos do tipo *goto* por todos os outros rótulos que aparecem na mesma função
SRSR	Troca cada comando de uma função por todos os comando *return* que existem na mesma função
STRI	É projetado para garantir que a condição de cada comando *if* seja avaliado pelo menos uma vez pelo ramo verdadeiro e pelo ramo falso
VDTR	Requer valores negativos, positivos e zero para cada referência escalar
VSCR	São responsáveis pela mutação de referência e componentes de uma estrutura que são substituídos por referências aos demais componentes da mesma estrutura, respeitando os tipos de componentes
VTWD	Troca referência escalar pelo seu valor sucessor e predecesssor
VGAR	Caso se tenha duas variáveis a e b, um vetor de caracteres e um ponteiro para inteiros respectivamente, a não pode ser substituído por b e vice-versa
VGPR	Os tipos são preservados, ou seja, assume-se que a variável do programa original e a variável que substituirá a variável original são ponteiros do mesmo tipo
VGSR	Troca cada referência escalar por todas as variáveis escalares locais e globais do programa
VGTR	Os tipos são preservados, caso se tenha duas estruturas de diferentes tipos elas não serão trocadas entre si para a criação do mutante
VLAR	Os tipos são preservados, ou seja, caso se tenha duas

	variáveis a e b, um vetor de caracteres e um ponteiro para inteiros respectivamente, a não pode ser substituído por b e vice-versa
VLPR	Os tipos são preservados, ou seja, assume-se que a variável do programa original e a variável que substituirá a original, são ponteiros do mesmo tipo
VLSR	Troca cada referência escalar por todas as variáveis escalares locais e globais do programa.
VLTR	Os tipos são preservados, caso se tenha duas estruturas de diferentes tipos elas não serão trocadas entre si para criação do mutante.
VSCR	São responsáveis pela mutação de referencia e componentes de uma estrutura que são substituídos por referencias aos demais componentes da mesma estrutura, respeitando os tipos de componentes.

Apêndice H:

Análise estística

Programa *Calcdate* - Casos de Testes Requeridos

Análise de variância - Programa CALCDATE - Mutantes Gerados - Casos de Testes Requeridos
16:38 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALCDATE - Mutantes Gerados - Casos de Testes Requeridos
16:38 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	21442.59866518	10721.29933259	449.30	0.0001
Error	57	1360.13466815	23.86201172		
Corrected Total	59	22802.73333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.940352	8.656011	4.88487581	56.43333333

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	9.99866518	9.99866518	0.42	0.5200
BLOCOS	1	21432.60000000	21432.60000000	898.19	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	9.99866518	9.99866518	0.42	0.5200
BLOCOS	1	21432.60000000	21432.60000000	898.19	0.0001

Análise de variância - Programa CALCDATE - Mutantes Gerados - Casos de Testes Requeridos
16:38 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 23.86201
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 2.5257

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	75.333	30	1
B	37.533	30	2

Análise de variância - Programa CALCDATE - Mutantes Gerados - Casos de Testes Requeridos

16:38 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 23.86201

Number of Means 2
Critical Range 2.526

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	75.333	30	1
B	37.533	30	2

Programa *Calcdate* - Mutantes Equivalentes

Análise de variância - Programa CALCDATE - Mutantes Gerados - Mutantes Equivalentes
16:36 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALCDATE - Mutantes Gerados - Mutantes Equivalentes
16:36 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	2880968.18146088	1440484.09073044	65.27	0.0001
Error	57	1258057.55187245	22071.18512057		
Corrected Total	59	4139025.73333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.696050	29.94834	148.56374093	496.06666667

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	7686.51479422	7686.51479422	0.35	0.5574
BLOCOS	1	2873281.66666667	2873281.66666667	130.18	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	7686.51479422	7686.51479422	0.35	0.5574
BLOCOS	1	2873281.66666667	2873281.66666667	130.18	0.0001

Análise de variância - Programa CALCDATE - Mutantes Gerados - Mutantes Equivalentes
16:36 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 22071.19
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 76.815

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	714.90	30	1
B	277.23	30	2

Análise de variância - Programa CALCDATE - Mutantes Gerados - Mutantes Equivalentes
16:36 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 22071.19

Number of Means 2
Critical Range 76.81

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	714.90	30	1
B	277.23	30	2

Programa *Calcdate* - Escore de Cruzamento

Análise de variância - Programa CALCDATE - Escore de Cruzamento
16:23 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALCDATE - Escore de Cruzamento
16:23 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	0.27805672	0.13902836	24.35	0.0001
Error	57	0.32547223	0.00571004		
Corrected Total	59	0.60352895			

R-Square	C.V.	Root MSE	SESSAO Mean
0.460718	9.966411	0.07556480	0.75819473

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	0.00440355	0.00440355	0.77	0.3835
BLOCOS	1	0.27365317	0.27365317	47.92	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	0.00440355	0.00440355	0.77	0.3835
BLOCOS	1	0.27365317	0.27365317	47.92	0.0001

Análise de variância - Programa CALCDATE - Escore de Cruzamento
16:23 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 0.00571
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 0.0391

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	0.817401	30	1
B	0.652488	30	2

16:23 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 0.00571

Number of Means 2
Critical Range .03907

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	0.817401	30	1
B	0.652488	30	2

Programa *Calen* - Casos de Testes Requeridos

Análise de variancia - Programa CALEN - Mutantes Gerados - Casos de Testes Requeridos
16:31 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variancia - Programa CALEN - Mutantes Gerados - Casos de Testes Requeridos
16:31 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	5571.81861327	2785.90930664	14.89	0.0001
Error	57	10664.91472006	187.10376702		
Corrected Total	59	16236.73333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.343161	60.61413	13.67858790	22.56666667

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	2136.55194661	2136.55194661	11.42	0.0013
BLOCOS	1	3435.26666667	3435.26666667	18.36	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	2136.55194661	2136.55194661	11.42	0.0013
BLOCOS	1	3435.26666667	3435.26666667	18.36	0.0001

Análise de variancia - Programa CALEN - Mutantes Gerados - Casos de Testes Requeridos
16:31 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 187.1038
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 7.0725

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	30.133	30	1
B	15.000	30	2

Análise de variancia - Programa CALEN - Mutantes Gerados - Casos de Testes Requeridos
16:31 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 187.1038

Number of Means 2
Critical Range 7.072

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	30.133	30	1
B	15.000	30	2

Programa *Calen* – Mutantes Equivalentes

Análise de variância - Programa CALEN - Mutantes Gerados - Mutantes Equivalentes
17:01 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALEN - Mutantes Gerados - Mutantes Equivalentes
17:01 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	3408951.38446422	1704475.69223211	800.04	0.0001
Error	57	121437.19886911	2130.47717314		
Corrected Total	59	3530388.58333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.965602	18.61799	46.15709234	247.91666667

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	21250.56779755	21250.56779755	9.97	0.0025
BLOCOS	1	3387700.81666667	3387700.81666667	1590.11	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	21250.56779755	21250.56779755	9.97	0.0025
BLOCOS	1	3387700.81666667	3387700.81666667	1590.11	0.0001

Análise de variância - Programa CALEN - Mutantes Gerados - Mutantes Equivalentes
17:01 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 2130.477
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 23.865

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	485.53	30	1
B	10.30	30	2

Análise de variância - Programa CALEN - Mutantes Gerados - Mutantes Equivalentes
17:01 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 2130.477

Number of Means 2
Critical Range 23.86

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	485.53	30	1
B	10.30	30	2

Programa *Calen* – Escore de Cruzamento

Análise de variância - Programa CALEN - Escore de Cruzamento

16:54 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALEN - Escore de Cruzamento
16:54 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	2.08615539	1.04307770	4117.06	0.0001
Error	57	0.01444123	0.00025335		
Corrected Total	59	2.10059662			

R-Square	C.V.	Root MSE	SESSAO Mean
0.993125	2.253657	0.01591713	0.70627968

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	0.00161847	0.00161847	6.39	0.0143
BLOCOS	1	2.08453693	2.08453693	8227.74	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	0.00161847	0.00161847	6.39	0.0143
BLOCOS	1	2.08453693	2.08453693	8227.74	0.0001

Análise de variância - Programa CALEN - Escore de Cruzamento
16:54 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 0.000253
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 0.0082

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	0.892673	30	1
B	0.519887	30	2

Análise de variância - Programa CALEN – Escore de Cruzamento
16:54 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 0.000253

Number of Means 2
Critical Range .008230

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	0.892673	30	1
B	0.519887	30	2

Programa *Calend* – Casos de Testes Requeridos

Análise de variância - Programa CALEND - Mutantes Gerados - Casos de Testes Requeridos
09:34 Monday, September 19, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa CALEND - Mutantes Gerados - Casos de Testes Requeridos
09:34 Monday, September 19, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	103.94705228	51.97352614	8.80	0.0005
Error	57	336.78628105	5.90853125		
Corrected Total	59	440.73333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.235850	9.633079	2.43074706	25.23333333

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	2.54705228	2.54705228	0.43	0.5141
BLOCOS	1	101.40000000	101.40000000	17.16	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	2.54705228	2.54705228	0.43	0.5141
BLOCOS	1	101.40000000	101.40000000	17.16	0.0001

Análise de variância - Programa CALEND - Mutantes Gerados - Casos de Testes Requeridos
09:34 Monday, September 19, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 5.908531
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 1.2568

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	26.5333	30	1
B	23.9333	30	2

```
Análise de variância - Programa CALEND - Mutantes Gerados - Casos de Testes Requeridos
                                                          09:34 Monday, September 19, 2005

                        General Linear Models Procedure

                Duncan's Multiple Range Test for variable: SESSAO

        NOTE: This test controls the type I comparisonwise error rate, not the
              experimentwise error rate

                     Alpha= 0.05  df= 57  MSE= 5.908531

                            Number of Means     2
                            Critical Range  1.257


        Means with the same letter are not significantly different.

              Duncan Grouping              Mean      N  BLOCOS

                           A           26.5333     30  1

           B           23.9333     30  2
```

Programa *Calend* – Mutantes Equivalentes

```
Análise de variância - Programa CALEND - Mutantes Gerados - Mutantes Equivalentes
                                                          09:45 Monday, September 19, 2005

                        General Linear Models Procedure
                            Class Level Information

                           Class    Levels    Values

                           BLOCOS        2    1 2

                        Number of observations in data set = 60


Análise de variância - Programa CALEND - Mutantes Gerados - Mutantes Equivalentes
                                                          09:45 Monday, September 19, 2005

                        General Linear Models Procedure

Dependent Variable: SESSAO

Source                   DF          Sum of Squares             Mean Square  F Value    Pr > F

Model                     2          359157.84530960         179578.92265480    69.24    0.0001

Error                    57          147839.88802373           2593.68224603

Corrected Total          59          506997.73333333

                  R-Square                     C.V.                Root MSE        SESSAO Mean

                  0.708401                10.98222             50.92820678       463.73333333


Source                   DF                Type I SS             Mean Square  F Value    Pr > F

REP                       1             949.57864294             949.57864294     0.37    0.5475
BLOCOS                    1          358208.26666667          358208.26666667   138.11    0.0001

Source                   DF             Type III SS             Mean Square  F Value    Pr > F

REP                       1             949.57864294             949.57864294     0.37    0.5475
BLOCOS                    1          358208.26666667          358208.26666667   138.11    0.0001


Análise de variância - Programa CALEND - Mutantes Gerados - Mutantes Equivalentes
                                                          09:45 Monday, September 19, 2005

                        General Linear Models Procedure

         Tukey's Studentized Range (HSD) Test for variable: SESSAO

        NOTE: This test controls the type I experimentwise error rate, but generally
              has a higher type II error rate than REGWQ.

                     Alpha= 0.05  df= 57  MSE= 2593.682
                Critical Value of Studentized Range= 2.832
                  Minimum Significant Difference= 26.332

        Means with the same letter are not significantly different.

              Tukey Grouping              Mean      N  BLOCOS

                           A           541.00     30  1
```

```
                                B            386.47     30  2


        Análise de variância - Programa CALEND - Mutantes Gerados - Mutantes Equivalentes
                                                              09:45 Monday, September 19, 2005

                            General Linear Models Procedure

                     Duncan's Multiple Range Test for variable: SESSAO

            NOTE: This test controls the type I comparisonwise error rate, not the
                  experimentwise error rate

                         Alpha= 0.05  df= 57  MSE= 2593.682

                               Number of Means     2
                               Critical Range  26.33

            Means with the same letter are not significantly different.

                 Duncan Grouping              Mean      N  BLOCOS

                                A            541.00     30  1

                                B            386.47     30  2
```

Programa *Calend* – Escore de Cruzamento

```
                        Análise de variância - Programa CALEND - Escore de Cruzamento
                                                              09:53 Monday, September 19, 2005

                                  General Linear Models Procedure
                                      Class Level Information

                                     Class    Levels    Values

                                     BLOCOS        2    1 2


                                Number of observations in data set = 60


                        Análise de variância - Programa CALEND - Escore de Cruzamento
                                                              09:53 Monday, September 19, 2005

                                  General Linear Models Procedure

        Dependent Variable: SESSAO

        Source                   DF          Sum of Squares             Mean Square  F Value    Pr > F

        Model                     2              2.01993276              1.00996638   236.07    0.0001

        Error                    57              0.24385875              0.00427822

        Corrected Total          59              2.26379150

                          R-Square                    C.V.                Root MSE         SESSAO Mean

                          0.892279                9.321666              0.06540813          0.70167855


        Source                   DF               Type I SS             Mean Square  F Value    Pr > F

        REP                       1              0.00917645              0.00917645     2.14    0.1485
        BLOCOS                    1              2.01075630              2.01075630   470.00    0.0001

        Source                   DF             Type III SS             Mean Square  F Value    Pr > F

        REP                       1              0.00917645              0.00917645     2.14    0.1485
        BLOCOS                    1              2.01075630              2.01075630   470.00    0.0001


                    Análise de variância - Programa CALEND - Escore de Cruzamento
                                                              09:53 Monday, September 19, 2005

                                  General Linear Models Procedure

                     Tukey's Studentized Range (HSD) Test for variable: SESSAO

            NOTE: This test controls the type I experimentwise error rate, but generally
                  has a higher type II error rate than REGWQ.

                               Alpha= 0.05  df= 57  MSE= 0.004278
                          Critical Value of Studentized Range= 2.832
                            Minimum Significant Difference= 0.0338

            Means with the same letter are not significantly different.

                 Tukey Grouping              Mean      N  BLOCOS

                                A            0.88474     30  1
```

```
                    B           0.51861     30  2


        Análise de variância - Programa CALEND - Escore de Cruzamento
                                                    09:53 Monday, September 19, 2005

                  General Linear Models Procedure

         Duncan's Multiple Range Test for variable: SESSAO

        NOTE: This test controls the type I comparisonwise error rate, not the
              experimentwise error rate

                Alpha= 0.05  df= 57  MSE= 0.004278

                     Number of Means       2
                     Critical Range  .03382

        Means with the same letter are not significantly different.

             Duncan Grouping              Mean      N  BLOCOS

                         A           0.88474     30  1

                         B           0.51861     30  2
```

Programa *Cal* – Casos de Testes Requeridos

```
      Análise de variância - Programa CAL - Mutantes Gerados - Casos de Testes Requeridos
                                                    16:41 Sunday, September 11, 2005

                        General Linear Models Procedure
                            Class Level Information

                           Class    Levels    Values

                           BLOCOS        2    1 2

                        Number of observations in data set = 60


      Análise de variância - Programa CAL - Mutantes Gerados - Casos de Testes Requeridos
                                                    16:41 Sunday, September 11, 2005

                        General Linear Models Procedure

Dependent Variable: SESSAO

Source                  DF          Sum of Squares             Mean Square   F Value     Pr > F

Model                    2          43692.86878013          21846.43439006     76.08     0.0001

Error                   57          16367.98121987            287.15756526

Corrected Total         59          60060.85000000

                  R-Square                  C.V.                Root MSE          SESSAO Mean

                  0.727477              33.92537            16.94572410          49.95000000


Source                  DF               Type I SS             Mean Square   F Value     Pr > F

REP                      1             6.85211346             6.85211346      0.02     0.8778
BLOCOS                   1          43686.01666667          43686.01666667    152.13     0.0001

Source                  DF             Type III SS             Mean Square   F Value     Pr > F

REP                      1             6.85211346             6.85211346      0.02     0.8778
BLOCOS                   1          43686.01666667          43686.01666667    152.13     0.0001


     Análise de variância - Programa CAL - Mutantes Gerados - Casos de Testes Requeridos
                                                    16:41 Sunday, September 11, 2005

                        General Linear Models Procedure

         Tukey's Studentized Range (HSD) Test for variable: SESSAO

        NOTE: This test controls the type I experimentwise error rate, but generally
              has a higher type II error rate than REGWQ.

                    Alpha= 0.05  df= 57  MSE= 287.1576
                  Critical Value of Studentized Range= 2.832
                    Minimum Significant Difference= 8.7618

        Means with the same letter are not significantly different.

             Tukey Grouping              Mean      N  BLOCOS

                         A            76.933     30  1
```

```
                                B            22.967     30  2


      Análise de variância - Programa CAL - Mutantes Gerados - Casos de Testes Requeridos
                                                              16:41 Sunday, September 11, 2005

                          General Linear Models Procedure

                  Duncan's Multiple Range Test for variable: SESSAO

        NOTE: This test controls the type I comparisonwise error rate, not the
              experimentwise error rate

                         Alpha= 0.05  df= 57  MSE= 287.1576

                               Number of Means      2
                               Critical Range  8.762

        Means with the same letter are not significantly different.

              Duncan Grouping              Mean      N  BLOCOS

                                A            76.933     30  1

           B            22.967     30  2
```

Programa *Cal* – Mutantes Equivalentes

```
                  Análise de variância - Programa CAL - Mutantes Gerados - Mutantes Equivalentes
                                                              16:40 Sunday, September 11, 2005

                                  General Linear Models Procedure
                                      Class Level Information

                                     Class    Levels    Values

                                     BLOCOS        2    1 2

                                Number of observations in data set = 60


                    Análise de variância - Programa CAL - Mutantes Gerados - Mutantes Equivalentes
                                                              16:40 Sunday, September 11, 2005

                                  General Linear Models Procedure

Dependent Variable: SESSAO

Source                   DF          Sum of Squares             Mean Square   F Value     Pr > F

Model                     2          356978.69091583          178489.34545792    122.62     0.0001

Error                    57           82967.64241750            1455.57267399

Corrected Total          59          439946.33333333

                   R-Square                     C.V.               Root MSE          SESSAO Mean

                   0.811414                15.27097            38.15196815          249.83333333


Source                   DF                Type I SS             Mean Square   F Value     Pr > F

REP                       1             5.62424917             5.62424917      0.00     0.9507
BLOCOS                    1          356973.06666667          356973.06666667    245.25     0.0001

Source                   DF             Type III SS             Mean Square   F Value     Pr > F

REP                       1             5.62424917             5.62424917      0.00     0.9507
BLOCOS                    1          356973.06666667          356973.06666667    245.25     0.0001


      Análise de variância - Programa CAL - Mutantes Gerados - Mutantes Equivalentes
                                                              16:40 Sunday, September 11, 2005

                          General Linear Models Procedure

                  Tukey's Studentized Range (HSD) Test for variable: SESSAO

        NOTE: This test controls the type I experimentwise error rate, but generally
              has a higher type II error rate than REGWQ.

                         Alpha= 0.05  df= 57  MSE= 1455.573
                    Critical Value of Studentized Range= 2.832
                      Minimum Significant Difference= 19.726

        Means with the same letter are not significantly different.

              Tukey Grouping              Mean      N  BLOCOS

                                A           326.967     30  1
```

```
                    B           172.700     30  2


Análise de variância - Programa CAL - Mutantes Gerados - Mutantes Equivalentes
                                                    16:40 Sunday, September 11, 2005

                    General Linear Models Procedure

          Duncan's Multiple Range Test for variable: SESSAO

     NOTE: This test controls the type I comparisonwise error rate, not the
           experimentwise error rate

                Alpha= 0.05  df= 57  MSE= 1455.573

                     Number of Means      2
                     Critical Range  19.73


     Means with the same letter are not significantly different.

           Duncan Grouping              Mean      N  BLOCOS

                        A           326.967     30  1

                        B           172.700     30  2
```

Programa *Cal* – Escore de Cruzamento

```
              Análise de variância - Programa CALCDATE - Escore de Cruzamento
                                                    16:23 Sunday, September 11, 2005

                    General Linear Models Procedure
                        Class Level Information

                       Class    Levels    Values

                       BLOCOS        2    1 2


                    Number of observations in data set = 60


              Análise de variância - Programa CALCDATE - Escore de Cruzamento
                                                    16:23 Sunday, September 11, 2005

                    General Linear Models Procedure

Dependent Variable: SESSAO

Source                   DF          Sum of Squares             Mean Square   F Value     Pr > F

Model                     2              0.27805672              0.13902836     24.35     0.0001

Error                    57              0.32547223              0.00571004

Corrected Total          59              0.60352895

                  R-Square                     C.V.               Root MSE          SESSAO Mean

                  0.460718                9.966411              0.07556480           0.75819473


Source                   DF               Type I SS             Mean Square   F Value     Pr > F

REP                       1              0.00440355              0.00440355      0.77     0.3835
BLOCOS                    1              0.27365317              0.27365317     47.92     0.0001

Source                   DF             Type III SS             Mean Square   F Value     Pr > F

REP                       1              0.00440355              0.00440355      0.77     0.3835
BLOCOS                    1              0.27365317              0.27365317     47.92     0.0001


           Análise de variância - Programa CALCDATE - Escore de Cruzamento

                                                    16:23 Sunday, September 11, 2005

                    General Linear Models Procedure

          Tukey's Studentized Range (HSD) Test for variable: SESSAO

     NOTE: This test controls the type I experimentwise error rate, but generally
           has a higher type II error rate than REGWQ.

                Alpha= 0.05  df= 57  MSE= 0.00571
              Critical Value of Studentized Range= 2.832
                Minimum Significant Difference= 0.0391

     Means with the same letter are not significantly different.
```

Tukey Grouping	Mean	N	BLOCOS
A	0.82573	30	1
B	0.69066	30	2

Análise de variância - Programa CALCDATE - Escore de Cruzamento
16:23 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 0.00571

Number of Means 2
Critical Range .03907

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	0.82573	30	1
B	0.69066	30	2

Programa *Jday* – Casos de Testes Requeridos

Análise de variância - Programa JDAY - Mutantes Gerados - Casos de Testes Requeridos
17:03 Sunday, September 11, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variância - Programa JDAY - Mutantes Gerados - Casos de Testes Requeridos
17:03 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	9142.27208009	4571.13604004	79.75	0.0001
Error	57	3267.06125324	57.31686409		
Corrected Total	59	12409.33333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.736725	33.89906	7.57079019	22.33333333

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	64.87208009	64.87208009	1.13	0.2919
BLOCOS	1	9077.40000000	9077.40000000	158.37	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	64.87208009	64.87208009	1.13	0.2919
BLOCOS	1	9077.40000000	9077.40000000	158.37	0.0001

Análise de variância - Programa JDAY - Mutantes Gerados - Casos de Testes Requeridos
17:03 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 57.31686
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 3.9145

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	34.633	30	1
B	10.033	30	2

Análise de variância - Programa JDAY - Mutantes Gerados - Casos de Testes Requeridos
17:03 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 57.31686

Number of Means 2
Critical Range 3.914

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	34.633	30	1
B	10.033	30	2

Programa *Jday* - Mutantes Equivalentes

Análise de variância - Programa JDAY - Mutantes Gerados - Mutantes Equivalentes

16:57 Sunday, September 11, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	19368.37122729	9684.18561364	3040.27	0.0001
Error	57	181.56210604	3.18530011		
Corrected Total	59	19549.93333333			

R-Square	C.V.	Root MSE	SESSAO Mean
0.990713	3.429995	1.78474091	52.03333333

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	0.30456062	0.30456062	0.10	0.7583
BLOCOS	1	19368.06666667	19368.06666667	6080.45	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	0.30456062	0.30456062	0.10	0.7583
BLOCOS	1	19368.06666667	19368.06666667	6080.45	0.0001

Análise de variância - Programa JDAY - Mutantes Gerados - Mutantes Equivalentes
16:57 Sunday, September 11, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 3.1853
Critical Value of Studentized Range= 2.832
Minimum Significant Difference= 0.9228

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	BLOCOS
A	70.0000	30	1
B	34.0667	30	2

Análise de variância - Programa JDAY - Mutantes Gerados - Mutantes Equivalentes
16:57 Sunday, September 11, 2005

General Linear Models Procedure

Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the experimentwise error rate

Alpha= 0.05 df= 57 MSE= 3.1853

Number of Means 2
Critical Range .9228

Means with the same letter are not significantly different.

Duncan Grouping	Mean	N	BLOCOS
A	70.0000	30	1
B	34.0667	30	2

Programa *Jday* – Escore de Cruzamento

Análise de variancia - Programa CALCDATE - Escore de Cruzamento
18:45 Sunday, September 5, 2005

General Linear Models Procedure
Class Level Information

Class	Levels	Values
BLOCOS	2	1 2

Number of observations in data set = 60

Análise de variancia – Programa CALCDATE - Escore de Cruzamento
18:45 Sunday, September 5, 2005

General Linear Models Procedure

Dependent Variable: SESSAO

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	2.83717123	1.41858561	1637.93	0.0001
Error	57	0.04936670	0.00086608		
Corrected Total	59	2.88653792			

R-Square	C.V.	Root MSE	SESSAO Mean
0.982898	3.967805	0.02942928	0.74170165

Source	DF	Type I SS	Mean Square	F Value	Pr > F
REP	1	0.00240253	0.00240253	2.77	0.1013
BLOCOS	1	2.83476869	2.83476869	3273.09	0.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
REP	1	0.00240253	0.00240253	2.77	0.1013
BLOCOS	1	2.83476869	2.83476869	3273.09	0.0001

Análise de variancia - Programa CALCDATE - Escore de Cruzamento
18:45 Sunday, September 5, 2005

General Linear Models Procedure

Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 57 MSE= 0.000866
Critical Value of Studentized Range= 2.832

```
                    Minimum Significant Difference= 0.0152

          Means with the same letter are not significantly different.

          Tukey Grouping              Mean      N  BLOCOS

                       A          0.959063     30  1

                       B          0.524340     30  2


             Análise de variancia - CALCDATE - Escore de Cruzamento
                                                          18:45 Sunday, September 5, 2005

                   General Linear Models Procedure

               Duncan's Multiple Range Test for variable: SESSAO

          NOTE: This test controls the type I comparisonwise error rate, not the
                experimentwise error rate

                 Alpha= 0.05  df= 57  MSE= 0.000866

                      Number of Means      2
                      Critical Range  .01522

          Means with the same letter are not significantly different.

          Duncan Grouping             Mean      N  BLOCOS

                       A          0.959063     30  1

                       B          0.524340     30  2
```

Programa *Dateplus* – Casos de Testes Requeridos

```
      Análise de variância - Programa DATEPLUS - Mutantes Gerados - Casos de Testes Requeridos
                                                          18:02 Friday, September 16, 2005

                          General Linear Models Procedure
                              Class Level Information

                             Class    Levels    Values

                             BLOCOS        2    1 2

                          Number of observations in data set = 60


      Análise de variância - Programa DATEPLUS - Mutantes Gerados - Casos de Testes Requeridos
                                                          18:02 Friday, September 16, 2005

                          General Linear Models Procedure

Dependent Variable: SESSAO

Source                   DF          Sum of Squares            Mean Square  F Value    Pr > F

Model                     2          22651.99121246          11325.99560623     6.72    0.0024

Error                    57          96109.65878754           1686.13436469

Corrected Total          59         118761.65000000

                   R-Square                  C.V.               Root MSE        SESSAO Mean

                   0.190735             232.6491            41.06256647        17.65000000

Source                   DF               Type I SS            Mean Square  F Value    Pr > F

REP                       1           9774.64121246           9774.64121246     5.80    0.0193
BLOCOS                    1          12877.35000000          12877.35000000     7.64    0.0077

Source                   DF            Type III SS            Mean Square  F Value    Pr > F

REP                       1           9774.64121246           9774.64121246     5.80    0.0193
BLOCOS                    1          12877.35000000          12877.35000000     7.64    0.0077


    Análise de variância - Programa DATEPLUS - Mutantes Gerados - Casos de Testes Requeridos
                                                          18:02 Friday, September 16, 2005

                          General Linear Models Procedure

           Tukey's Studentized Range (HSD) Test for variable: SESSAO

          NOTE: This test controls the type I experimentwise error rate, but generally
                has a higher type II error rate than REGWQ.

                        Alpha= 0.05  df= 57  MSE= 1686.134
```

```
                Critical Value of Studentized Range= 2.832
                  Minimum Significant Difference= 21.231

          Means with the same letter are not significantly different.

          Tukey Grouping              Mean      N  BLOCOS

                       A             32.30     30  1

                       B              3.00     30  2


  Análise de variância - Programa DATEPLUS - Mutantes Gerados - Casos de Testes Requeridos
                                                        18:02 Friday, September 16, 2005

                        General Linear Models Procedure

               Duncan's Multiple Range Test for variable: SESSAO

          NOTE: This test controls the type I comparisonwise error rate, not the
                experimentwise error rate

                     Alpha= 0.05  df= 57  MSE= 1686.134

                            Number of Means     2
                            Critical Range  21.23

          Means with the same letter are not significantly different.

               Duncan Grouping              Mean      N  BLOCOS

                       A             32.30     30  1

                       B              3.00     30  2
```

Programa *Dateplus* – Mutantes Equivalentes

```
          Análise de variância - Programa DATEPLUS - Mutantes Gerados - Mutantes Equivalentes
                                                        18:06 Friday, September 16, 2005

                                General Linear Models Procedure
                                    Class Level Information

                                   Class    Levels    Values

                                   BLOCOS        2    1 2

                                Number of observations in data set = 60


          Análise de variância - Programa DATEPLUS - Mutantes Gerados - Mutantes Equivalentes
                                                        18:06 Friday, September 16, 2005

                                General Linear Models Procedure

Dependent Variable: SESSAO

Source                  DF           Sum of Squares             Mean Square  F Value    Pr > F

Model                    2            375.00000000            187.50000000 99999.99    0.0001

Error                   57              0.00000000              0.00000000

Corrected Total         59            375.00000000

                  R-Square                     C.V.               Root MSE        SESSAO Mean

                  1.000000                        0                      0       393.50000000


Source                  DF              Type I SS             Mean Square  F Value    Pr > F

REP                      1              0.00000000              0.00000000       .         .
BLOCOS                   1            375.00000000            375.00000000 99999.99    0.0001

Source                  DF             Type III SS             Mean Square  F Value    Pr > F

REP                      1              0.00000000              0.00000000       .         .
BLOCOS                   1            375.00000000            375.00000000 99999.99    0.0001


        Análise de variância - Programa DATEPLUS - Mutantes Gerados - Mutantes Equivalentes
                                                        18:06 Friday, September 16, 2005

                        General Linear Models Procedure

          Tukey's Studentized Range (HSD) Test for variable: SESSAO

          NOTE: This test controls the type I experimentwise error rate, but generally
```

```
has a higher type II error rate than REGWQ.

                Alpha= 0.05  df= 57  MSE= 0
        Critical Value of Studentized Range= 2.832
             Minimum Significant Difference= 0

Means with the same letter are not significantly different.

      Tukey Grouping              Mean      N  BLOCOS

                   A             396.0     30  1

                   B             391.0     30  2
```

```
Análise de variância - Programa DATEPLUS - Mutantes Gerados - Mutantes Equivalentes
                                                    18:06 Friday, September 16, 2005

                    General Linear Models Procedure

          Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the
      experimentwise error rate

                Alpha= 0.05  df= 57  MSE= 0

                     Number of Means 2
                     Critical Range  0

Means with the same letter are not significantly different.

      Duncan Grouping             Mean      N  BLOCOS

                   A             396.0     30  1

                   B             391.0     30  2
```

Programa *Dateplus* – Escore de Cruzamento

```
                Análise de variância - Programa DATEPLUS - Escore de Cruzamento
                                                    18:09 Friday, September 16, 2005

                    General Linear Models Procedure
                        Class Level Information

                       Class    Levels    Values

                       BLOCOS        2    1 2

                    Number of observations in data set = 60
```

```
                Análise de variância - Programa DATEPLUS - Escore de Cruzamento
                                                    18:09 Friday, September 16, 2005

                    General Linear Models Procedure

Dependent Variable: SESSAO

Source                   DF          Sum of Squares             Mean Square  F Value    Pr > F

Model                     2              0.77863498              0.38931749  1454.02    0.0001

Error                    57              0.01526188              0.00026775

Corrected Total          59              0.79389686

                  R-Square                     C.V.                Root MSE        SESSAO Mean

                  0.980776                2.440579              0.01636314         0.67046138


Source                   DF               Type I SS             Mean Square  F Value    Pr > F

REP                       1              0.00088988              0.00088988     3.32    0.0735
BLOCOS                    1              0.77774510              0.77774510  2904.72    0.0001

Source                   DF             Type III SS             Mean Square  F Value    Pr > F

REP                       1              0.00088988              0.00088988     3.32    0.0735
BLOCOS                    1              0.77774510              0.77774510  2904.72    0.0001
```

```
                Análise de variância - Programa DATEPLUS - Escore de Cruzamento
                                                    18:09 Friday, September 16, 2005

                    General Linear Models Procedure

          Tukey's Studentized Range (HSD) Test for variable: SESSAO

NOTE: This test controls the type I experimentwise error rate, but generally
```

```
        has a higher type II error rate than REGWQ.

             Alpha= 0.05  df= 57  MSE= 0.000268
          Critical Value of Studentized Range= 2.832
            Minimum Significant Difference= 0.0085

Means with the same letter are not significantly different.

      Tukey Grouping              Mean      N  BLOCOS

                   A          0.784314     30  1

                   B          0.556609     30  2


   Análise de variância - Programa DATEPLUS - Escore de Cruzamento
                                                 18:09 Friday, September 16, 2005

             General Linear Models Procedure

     Duncan's Multiple Range Test for variable: SESSAO

NOTE: This test controls the type I comparisonwise error rate, not the
      experimentwise error rate

           Alpha= 0.05  df= 57  MSE= 0.000268

                Number of Means       2
                Critical Range  .008460

Means with the same letter are not significantly different.

     Duncan Grouping              Mean      N  BLOCOS

                   A          0.784314     30  1

                   B          0.556609     30  2
```